A Two-Tiered Approach for Decentralized Reasoning in Ambient Intelligence

José Viterbo, Markus Endler, and Gustavo Baptista, Pontifícia Universidade Católica do Rio de Janeiro

This article proposes a two-tiered model for decentralized reasoning in which two interacting parties—with access to different context information—perform a cooperative strategy to infer situations.

Ambient intelligence (AmI) is a multidisciplinary approach that aims to integrate innovative sensing, communication, and actuation technologies to create computer-mediated environments that support user activities with minimal user intervention. Essentially, an AmI system should be aware of people’s presence in the geographical space, perceive their needs, and be able to autonomously personalize and provide services that help them perform their tasks. Therefore, AmI applications and services must be capable of automatically responding to dynamic changes in the environment—such as a person entering or leaving a room, a steep temperature increase, or a new device connecting to the local network—with no or minimal human interference. Hence, applications executing in AmI systems are intrinsically context-aware; they must strongly rely on context data collected from sensors embedded in the environment and the user’s devices to trigger adaptations at different levels, such as in the wireless communication links, middleware and application services, or user interfaces.

AmI requires middleware support for software development and deployment that can integrate many different devices and sensors and build a programmable and autoconfigurable infrastructure. In general, these middleware systems provide services to store, distribute, and process context data collected from different sources as well as services for reasoning about context information. Reasoning is useful in dealing with the intrinsic imperfection and uncertainty of context data. It can detect possible errors, estimate missing values, and determine the context data’s quality and validity. Reasoning might also be used for inferring new, implicit context information, derived from lower-level context data that might be meaningful and relevant for many applications. Lastly, reasoning helps identify specific context-dependent situations, which consist of particular combinations of states of aggregated context data that are relevant for triggering actions.
or adaptations in applications or services.

This article focuses on the last purpose of reasoning: identifying application-specific situations. Developers might describe these situations apart from the application code, using rules (in formal frameworks) based on some type of logic. This approach entails great flexibility because it lets us easily modify previously defined rules to adapt applications to different domains and reuse available rules in different applications. Therefore, rule-based inference mechanisms are key to middleware systems that support the development and deployment of AmI services and applications.

In AmI environments, automated-reasoning operations might need to evaluate context data collected from distributed sources and stored in different devices since not all context data is usually readily available to all devices, as in ubiquitous systems. To address this, we propose a decentralized reasoning approach, in which we divide AmI services and applications into two main interacting parties: a user side and an ambient side. Reasoners running at each side have access to different context information, which is not shared with the other side. We implemented and tested a prototype middleware service that realizes this approach. Using an AmI scenario that exemplifies our reasoning service’s target environment, we discuss our two-tiered system model and explain our cooperative strategy for rule-based reasoning.

**UbiConference Scenario**

As a typical scenario to exemplify our approach, we consider a fictitious ubiquitous computing conference (UbiConference), where several researchers from different universities and companies gather to present and discuss their recent work. We assume that the conference comprises several technical sessions on subjects such as middleware, ambient intelligence, and so forth. In this setting, Silva is a lecturer and researcher who is participating as a general attendee.

UbiConference takes place in a convention center with several plenary rooms equipped with an infrastructure to provide the organizing committee and attendees with ambient services. The conference organizer (ConfOrg) service, which is part of this infrastructure, aims to provide context-aware functionalities such as alerting a program committee member if the session chair has not shown up in the corresponding room when the session is about to start. Moreover, a conference companion (ConfComp) application is available to be downloaded onto the participants’ notebooks when they register on the conference website. ConfComp helps the participants not only with their agendas during the event—notifying the participants about the beginning of presentations in which they might be interested—but also by identifying people with similar subject interests, thus stimulating the collaboration and social interactions among researchers at the event.

ConfComp interacts with ConfOrg to provide ubiquitous services tailored for each participant’s preferences. After installing ConfComp, participants are asked to provide detailed information about their affiliations and subjects of interest. Besides that, the participants’ notebooks run an indoor positioning service, capable of continuously determining their location. However, the participants’ locations and preferences are not disclosed to ConfOrg or the other attendees.

In such a system, the ambient services and the applications executing on the users’ devices personalize their functional behavior based on the context data available at the moment, but each has access to different parts of the overall context information. For instance, when Silva arrives at the UbiConference venue, the ConfOrg service detects that his notebook has connected to the local wireless network and automatically registers his presence at the conference. By then, Silva’s ConfComp gets the updated schedule of that day’s sessions, so he can select which to attend. From this moment on, whenever he is outside the room of a session that he selected and is about to start, ConfComp notifies him to hurry to the corresponding room. In between the sessions for which Silva expressed an interest, ConfComp might suggest that he attend another presentation where much of the audience shares similar interests.

**Two-Tiered Model**

In our system model, the user and ambient interacting parties in the reasoning process consist of the services, applications, and data that are available at each side. For several reasons, ranging from privacy to performance issues, some information might be available only at the user side, while some other information might be available only at the ambient side. For instance, on the user side, Silva’s notebook stores information about his affiliation, his subjects of interest, and the sessions he wants to attend, in addition to his devices’ resources and location. On the ambient side, the infrastructure stores information about activities and the environment, such as room assignment and each session’s status, the status of
devices (such as projectors in the session rooms), and the sound volume in the rooms. Both sides share a common context model, but store different context data. Figure 1 shows an example of some context information that might be available on each side at a given moment.

We consider that applications executing on the ambient infrastructure or on the user’s device rely on a rule-based reasoning service provided by the middleware to identify context-dependent relevant situations. As a means of describing context information, we adopt an ontology-based model. We describe situations using a Description Logics (DL) safe rule $R$, in which the free variables correspond to individuals that might be available at either side. These applications can query or subscribe to the reasoning service, providing a rule $R$ to be inferred. The result of the reasoning operation is a set of tuples $S$ containing individuals that satisfy the rule. We assume that there is no message loss in the communication among the system entities.

Hence, for a reasoning service running on either side to infer relevant situations based on rules provided by applications, we devise three possible patterns of interaction:

- **user-side reasoning**, if the reasoning is performed based only on context information available on the user side;
- **ambient-side reasoning**, if the reasoning is performed based only on context information available on the ambient side; or
- **cooperative reasoning**, when the reasoning is performed based on context information stored on both sides.

In user-side reasoning, applications executing on mobile devices might be interested in situations expressed by rules in which all the involved free variables represent context information available at the device. For example, in our scenario, the ConfComp device might be configured to warn Silva to recharge his notebook when he is waiting between sessions in the conference center lobby. The application needs to identify the situation described by Rule 1:

$$\text{hasEnergyLevel}(	ext{"Notebook-Silva," } ?c) \land \text{lessThan}(?c, \text{ "40%"}) \land \text{isLocatedIn}\left(\text{"Silva," \ "Lobby"}\right) \rightarrow \text{shouldRecharge}\left(\text{"Notebook-Silva"}\right)$$

This situation involves the device’s capabilities and the user’s location; hence, the desired situation could be inferred solely based on context data available at the device. The rule states that if Silva is located in the lobby and his mobile device’s battery has less than 40 percent energy, then he should recharge it. In this case, a reasoning service executing on the device could check when the rule can be triggered.

In contrast, if all context data necessary to find the result for a rule is available to the ambient infrastructure’s services, the reasoning operation can be performed entirely in the ambient-side reasoning. Revisiting our scenario, let us assume that an application at the ambient side is interested in the situation described by Rule 2:

$$\text{takesPlace}(?s, ?r) \land \text{hasStarted}(?s) \rightarrow \text{isBusy}(?r)$$

which states that if a conference session takes place in a given room and the session has already started, then the room is busy. In this case, a reasoning service running on the ambient infrastructure could infer the rule.

As a third possibility, a situation might be described by a rule containing context variables that refer to context information available both at the devices and in the ambient...
infrastructure. For instance, consider Rule 3:

\[
isLocatedIn(“Silva,” ?r) \land takesPlace (?s,?r) \land hasStarted(?s) \rightarrow isBusy(“Silva”)
\]

which states that if Silva is located in a room where a conference session is supposed to take place and this session has already started, then Silva is busy. As Figure 1 shows, the user’s location data is available only on the user side, while the information about the sessions is available only on the ambient side.

In this case, for a reasoning service executing in the ambient infrastructure to be able to infer the rule, it would have to collect and store all context data produced in that environment, both by the ambient infrastructure itself and by all the users’ devices, in a centralized way. Although there are usually no computational limitations for the ambient infrastructure to store or process large amounts of data, privacy issues might prevent users from disclosing personal information—such as location or personal preferences—to the ambient infrastructure. In this case, the device on the user side could collect and store all context information available on the ambient side that would be necessary for reasoning about the proposed rules. Following this approach, however, would entail another problem: the reasoning computation might be too heavy to be performed by a resource-limited mobile device.

We conclude that, for such types of rules, reasoning should be executed involving reasoning services on both sides through cooperative reasoning. In cooperative reasoning, both sides analyze the locally available context information, then combine the outcomes to produce an appropriate result. For this purpose, however, we must define a cooperation strategy in the reasoning operation.

**Cooperative Reasoning**

We propose a strategy in which a device reasoner running on the user side and an ambient reasoner running on the ambient side interact to infer situations described by rules involving context variables depending on data collected from different sources and stored at both sides. Together these reasoners perform cooperative reasoning. Each of these services, called cooperative reasoners, aggregate context information obtained from local context providers available at each side and execute the reasoning of rules submitted by applications running on either side. Because the interaction might occur on either start side, depending on which side the client application is running, we call the local reasoner the one executing on the side where the interaction begins and call the other the remote reasoner.

The client application might query the reasoner to get an immediate response \( S \) about a submitted rule \( R \) or might subscribe to be notified whenever the situation described by \( R \) holds. As such, this cooperation can involve either synchronous or asynchronous interaction. In any case, the cooperative reasoning process consists of the following steps:

1. Splitting the consequent of the rule into local and remote parts.
2. Performing the reasoning about the local part of the rule in the local knowledge base to find a preliminary result \( T \).
3. Forwarding the preliminary result \( T \) from the local reasoner to the remote reasoner.
4. Performing the reasoning about the remote part of the rule in the remote knowledge base, bounded by the preliminary result \( T \) to obtain \( S \).

The synchronous interaction starts when a client application submits a synchronous query to the local reasoner. After receiving a submission for rule \( R \), the reasoner parses it and splits the antecedent of the original rule into two parts: a local part, comprised of atoms that refer only to context information available at the local reasoner, and a remote part, comprised of the atoms that refer to context information available at the remote reasoner. In our model we assume that

- each reasoner knows whether an atom of the rule corresponds to context data available at its side, and
- each atom corresponds to context data available either on the device side or on the ambient side—that is, all predicates are valid.

The remote part is forwarded to the remote reasoner together with the preliminary result \( T \) obtained at the local reasoner. Finally, the remote reasoner infers the response \( S \), which is forwarded to the application.

The asynchronous interaction starts when a client application submits a rule \( R \) to the local reasoner as a subscription. As in the synchronous interaction, the original rule’s antecedent is split into a local and a remote part, which is forwarded to the remote reasoner together with the preliminary result \( T \). In the remote reasoner, the remote part is put into a list of subscriptions. Each time a change happens in context data that might affect the rule, the rule is selected so that the reasoner can perform another evaluation of the rule. If the rule is satisfied—that is, if a response \( S \) is found—it is forwarded
to the application. While the remote part is being monitored by the remote reasoner, changes in the context data might also happen on the local side, which might cause a change in the set of tuples $T$ that were previously forwarded to the remote reasoner. Therefore, this information is updated at the remote reasoner whenever a new set of tuples $T$ is found. (All the algorithms that comprise the cooperative reasoning process, describing both the synchronous and asynchronous interactions, are presented in detail in previous research.3)

**Context Data Stability**
A notification from the remote reasoner to the local reasoner about the result of a forwarded rule is based on the context data available at the remote reasoner at the moment it was generated and the latest set of tuples $T$ received from the local reasoner. Before the notification from the remote reasoner arrives at the local reasoner, however, there might have been a context data change at the local reasoner that caused a new update to be sent to the remote reasoner. In this case, the result $S$ received from the remote reasoner cannot be considered valid because it is based on a set of tuples $T$ that has changed.

To prevent the local reasoner from sending an inaccurate result to the client application, each time the local reasoner generates a new update, it receives an update number, which is sent to the remote reasoner together with the new set of tuples $T$. When the remote reasoner finds a set of values $S$ that satisfies a rule, it notifies the local reasoner, sending the result $S$, together with the update number of the latest update it received, letting the local reasoner check if the result corresponds to the latest update. After receiving this notification, the local reasoner sends the received result $S$ to the client application only if the number of the last update received by the remote reasoner matches the number of the last update sent by the local reasoner. Otherwise, the result received from the remote reasoner is ignored.

If there are frequent changes of the context data related with a rule $R$ at the local reasoner, however, the reasoners might never converge to find a response and notify the client application. Hence, this strategy is inadequate for reasoning with context data that are highly variable.

**Decentralized Reasoning Service**
We implemented the DRS as a prototype middleware service that realizes our proposed approach for decentralized reasoning, performing the cooperative reasoning process. We designed this rule-based inference service to be executed on the top of a communication and context provisioning middleware architecture for mobile collaboration.4 The middleware APIs were used to implement synchronous and asynchronous event-based communication, allowing the identification of client disconnections, but not supporting automatic reconnection. Moreover, the middleware supports DRS with services for collecting context data from context providers available in a specific domain, keeping an updated representation of the assembled data according to a valid context model (an ontology) and providing access to up-to-date context information.

To access ontology data and perform reasoning operations, DRS relies on KAON2, an Web Ontology Language (OWL) based ontology management and reasoning API developed by Boris Motik.3 Compared with other systems, KAON2 provides better performance for ontologies with rather small TBoxes, but large ABoxes—that is, ontologies with a large number of individuals and facts and a small number of classes and properties.6 We believe this is the case when using reasoners to infer AmI application-specific situations.

We also provided a DRS API for implementing client applications that use the inference services provided by DRS. Using this API, developers can easily create client applications that interact with a local DRS to check if a rule is satisfied (synchronously) or to post or remove a subscription. To show how DRS can be used to support the implementation and execution of a ubiquitous application, the performance evaluation of a prototype application are available in a Web extra (see www.lac.inf.puc-rio.br/drs/prototype.html).

**Performance Evaluation**
To test DRS’s performance, we created an ontology based on our conference example use case and measured the DRS server’s behavior under different simulated load conditions. This ontology contained the four classes Person, Activity, Subject, and Environment interrelated with the five binary properties isLocatedIn, wantsToAttend, isInterestedIn, isRelatedWith, and takesPlace, besides the unary property (subclass) isAboutToStart (which is not represented in Figure 1). Trying to mimic numbers that could possibly occur in a real scenario (such as a large conference), we generated numerous individuals and facts to create a large ABox. For example, at the ACM Symposium on Applied Computing (SAC) 2008, there were about 60 different sessions and the proceedings included approximately 1,400 keywords. Based on these data, we generated an ontology containing 2,170 individuals and 27,360 facts.
Our testbed consisted of a desktop PC with a Core 2 Duo 2.40-GHz processor, 4 GBytes of RAM, and Windows 7 operating system to execute the ambient-side DRS and a netbook with an Atom 1.60 GHz processor, 1 GByte of RAM, and the Windows XP SP2 operating system to execute the user-side DRS and an application client. These computers were interconnected through an IEEE 802.11 wireless network.

In a first experiment, we measured the response time—the overall time from when a context data change message is sent to the ambient-side DRS until an application client receives the response. We simulated the load of the ambient-side DRS creating a variable number of subscriptions—100, 200, 300, 400, or 500—defined by Rule 4:

\[
\text{wantsToAttend}("Silva", ?s) \land \text{takesPlace}(?s, ?r) \land \text{isAboutToStart}(?s) \rightarrow \text{shouldGoToRoom}("Silva", ?r)
\]

Moreover, we simulated context changes being sent to the DRS every 50 milliseconds, causing the server to constantly work to check all the subscriptions involved with the context changes informed.

As expected, increasing the number of subscriptions also caused the system’s response time to increase. In the best case we measured approximately 0.8 seconds, and in the worst case, we measured approximately 10.4 seconds (see Figure 2). This result, which is acceptable for our target scenario, proves the applicability of the cooperative reasoning process in real applications.

In a second experiment, we compared DRS’s decentralized reasoning approach with both a centralized and a simple peer-to-peer (P2P) approach to evaluate the inference Rule 3 for the same ontology and testbed used in the first experiment. We measured the communication traffic between the mobile device and the network, simulating different configurations. Distributed context providers were simulated by programs running in the background and generating context change messages every 5 seconds.

To represent the mobile users, we implemented location providers, varying the \text{isLocatedIn} property. To represent the ambient context changes, we implemented activity status providers, indicating that a conference session \text{isAboutToStart}, \text{hasStarted}, or \text{hasFinished}. Although we executed one location provider instance on the netbook (LP1), all other context providers—499 location provider instances (LP2 to LP500) and one activity status provider instance (ASP)—were deployed on the stationary machine.

For the decentralized reasoning approach, we set LP1 to send context-change messages regarding the user’s location to the user-side DRS on the netbook and ASP to send context-change messages to the ambient-side DRS on the stationary machine. The other location provider instances were ignored. For simulating and evaluating the centralized reasoning approach, a DRS server executing on the ambient side was set to be the sole reasoner, receiving all context change messages from LP1 to LP500 and ASP.

Because we could not find performance results of any related system implementing the P2P reasoning approach, we decided to simulate a simple P2P system, where each user-side DRS was responsible for performing all inferences locally using context updates received from all other peers. Thus, for setting up the P2P configuration, we configured LP1 to LP500 and ASP to periodically broadcast context-change messages to the user-side DRS.

For these three approaches we measured the communication traffic at the netbook, considering both received and sent context-change messages and the messages exchanged between the DRS services, during a simulation period of 5 minutes. We also verified the memory footprint of the reasoner executing on that device. The measured values in Table 1 show

![Figure 2. Response time measured for the decentralized reasoning service (DRS) prototype executing decentralized reasoning.](image-url)

<table>
<thead>
<tr>
<th>Values</th>
<th>Centralized</th>
<th>Decentralized</th>
<th>Peer to peer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communication traffic</td>
<td>790 bytes</td>
<td>81.7 bytes</td>
<td>303.5 KBytes</td>
</tr>
<tr>
<td></td>
<td>per sec.</td>
<td>per sec.</td>
<td></td>
</tr>
<tr>
<td>Memory footprint</td>
<td>–</td>
<td>20.2 KBytes</td>
<td>23.5 KBytes</td>
</tr>
</tbody>
</table>
DISTRIBUTED REASONING

The Authors

José Viterbo is an adjunct professor in the Department of Informatics at Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio). His research interests include ubiquitous computing, mobile collaboration, and social computing. Viterbo has a DSc in computer science from PUC-Rio. Contact him via www.lac.inf.puc-rio.br/~viterbo.

Markus Endler is an associate professor in the Department of Informatics at PUC-Rio. His research interests include distributed algorithms and systems, ubiquitous computing, and cloud computing. Endler has a DSc in computer science from the Technical University in Berlin and the title habilitated doctor from the University of São Paulo. Contact him via www.inf.puc-rio.br/~endler.

Gustavo Baptista is a PhD candidate in the Department of Informatics at PUC-Rio. His research interests include middleware for mobile networks and cloud computing. Baptista has a MSc in computer science from PUC-Rio. Contact him via www.lac.inf.puc-rio.br/~gustavo.

that our decentralized approach minimizes the communication with the mobile device and, due to a smaller ontology, requires less memory at the user side.

Performance tests showed that DRS offers satisfactory response times. However, the scalability of the current implementation was constrained by the increasing use of memory and the communication overhead, when the number of subscriptions grew. In our tests, memory overflow limited the number of subscriptions we could simulate to approximately 700 clients, and the performance was greatly affected for more than 400 subscriptions.

Despite the limitation of the current DRS implementation, we believe that this work paves the way toward a strategy and protocol for decentralized reasoning. Nevertheless, we think that the scalability of the DRS can be considerably improved. In our current prototype, we have yet to optimize the use of memory, keeping large data structures stored in memory because we focused on improving the execution time performance. However, in our tests, we concluded that the time consumed with the reasoning operation was not critical, indicating that the implementation might be revised, prioritizing a reduction of the use of memory. The communication overhead is another factor that limits our implementation's scalability because it increases the response time when there are many subscriptions. As such, the protocol might be improved to reduce the message exchange between the local and remote reasoners. Instead of sending updates every time a single context variable changes, we could reduce the frequency of update messages by aggregating updates on several context variables over a period of time. Of course, this solution might also increase the system's reasoning latency, but this might be acceptable for most AmI applications, such as the UbiConference scenario we describe here.

The current version of DRS depends on a communication middleware layer ensuring reliable message delivery, the main reason being that the DRS cooperative protocol does not implement message acknowledgments. However, one of our next steps will be to port DRS to a publish-subscribe system that we have already developed (MD-ECI). This system implements reliable message delivery and can handle client disconnections and dynamic changes of its IP addresses. In addition, because the use of smart phones is fundamental in ubiquitous systems, porting our implementation to execution environments such as Google’s Android is a high-priority task. Furthermore, it is important to make a new step toward a more general distributed reasoning scenario. As such, we intend to study how to extend the proposed strategy to allow the inference of rules in scenarios where the context data is divided in more than two tiers, such as for multipeer coordination.

Acknowledgments

This work was partially funded by projects Conselho Nacional de Desenvolvimento Científico e Tecnológico (Brazilian Council of Technological and Scientific Development, CNPq) 55.2068/02-2 European Stadium and Safety Management Association (ESSMA) and 479824/04-5 (Ed. Universal).

References