A Context-Aware Middleware for Multimodal Dialogue Applications with Context Tracing

Manh-Thang Tran  
Department of Informatics  
University of Fribourg  
Chemin du Musée 3  
1700 Fribourg, Switzerland  
tranmanh.thang@unifr.ch

Béat Hirsbrunner  
Department of Informatics  
University of Fribourg  
Chemin du Musée 3  
1700 Fribourg, Switzerland  
beat.hirsbrunner@unifr.ch

Michèle Courant  
Department of Informatics  
University of Fribourg  
Chemin du Musée 3  
1700 Fribourg, Switzerland  
michele.courant@unifr.ch

ABSTRACT
This paper presents a context-aware middleware for multimodal dialogue applications. The middleware has the context tracing feature, which is the possibility of the middleware to explain why and how a situation occurs (or occurred). The middleware consists of several agents communicating with each other and an ontology is used to describe various concepts such as resources, situations, plans, structure of context history, and data to be exchanged between the agents.

Categories and Subject Descriptors
D.2.10 [Software Engineering]: Design—Methodologies; I.2.11 [Artificial Intelligence]: Distributed Artificial Intelligence

General Terms
Design

Keywords
Middleware, multimodal dialogue, ontology, context tracing

1. INTRODUCTION
The wide spreading of computing devices in everyday environments and the resulting pervasion of artificial intelligence in the physical space - usually designated as pervasive or ubiquitous computing - bring on new challenges to computing methodologies. Devices and software agents are structured around context-awareness, a feature of computing systems that are able to perceive relevant changes in their environment in order to act accordingly. Much research has shown that the development of an application in a pervasive environment could be facilitated using an available middleware [3], [7]. The middleware may provide support for most of the tasks involved in dealing with context-awareness.

Multimodal dialogue applications in pervasive environments allow humans to interact with their systems in natural ways using speech, gesture or facial expression. For this kind of applications, context-awareness appears very often in two types. The first type is how the system understands the user’s request in order to implement suitable tasks. For example, consider in a Smart-home environment, when a user says: “Switch it on please” while pointing his finger to a light, the system will switch on the light. In order to do this, the system must somehow capture the user’s speech and gesture in order to react accordingly. The second type is how the system may explain to the user the cause of a certain situation. For example, in case a television automatically starts to show a certain movie, the user may ask the system: “Why do you show this movie?”. To answer this, the system must have an important feature which we call context tracing. It is the possibility of the system to explain why and how a situation occurs (or occurred) using causal relations between situations. A past situation is considered as a cause of the current one - the consequence if its existence has activated some operations that lead to the existence of the current one. The causes that the television shows the movie include “It is 8pm” and the existence of a plan: “today, if it is 8pm then show that movie”. The system can answer the user: “Because it is 8pm and there exists a plan such that today, if it is 8pm then I must show that movie”. When developing a multimodal dialogue application in a pervasive environment, it is important that the developers have to take these two types of context-awareness into account.

To develop a multimodal dialogue application, the developers face the problem about the heterogeneity of components and technologies for capturing user input and presenting system output. For example, spoken dialogue systems uses following kinds of components for this task: speech recognition, speech synthesis, natural language understanding and generation. These components might be written in different programming languages or running on different platforms, using different techniques. When one of these components changes, developers have to re-design the application. The existence of a middleware could solve this problem. It uses a uniform structure to represent all kinds of user input and system output. If an user input agent (e.g., microphone with a software module) has captured user data (no matter what technology it uses for capturing), the agent
has to represent this data in that uniform structure before sending it to the middleware. In the other way around, a system output agent (e.g., loudspeaker with a software module) only accepts data from the middleware if this data is represented in that uniform structure. The complexity of capturing user input and presenting system output is delegated to agents. The development of an application becomes the task of developing the dialogue management part only, using the data represented in that uniform structure. The middleware can also have the context tracing feature.

In the CAMA project [1], we aim to develop a middleware (also called the CAMA middleware) for multimodal dialogue applications in pervasive environments. An application includes one or several dialogue handling modules (or another name, discourse processing modules) built on top of the middleware. Like the other middlewares for pervasive environments, the CAMA middleware can gather low-level contextual information from sensor signals and human behaviors (speech, gesture, and facial expression), infer high-level contextual information from that low-level data, and fulfill the implementation of services. Furthermore, it has the following features:

- Planning: Simple plans can be defined (e.g., via user-system dialogues) and implemented by the middleware.
- Context tracing: All causal relations between situations are maintained.
- Context-searching: It is the possibility of the middleware to identify a situation based on some attributes (e.g., to identify a research paper that the user downloaded two days ago, in between 9h and 11h, at the computer 002).

The CAMA middleware consists of several agents communicating with each other and an ontology is used to describe various concepts such as resources, situations, plans, structure of context history, and data to be exchanged between the agents. The use of ontology supports semantic interoperability between entities and context-awareness.

This paper describes the CAMA middleware structure with the focus of how context tracing is performed. The paper’s content is organized as follows. In section 2, we present some research works related to context-aware middleware and multimodal dialogue applications. The CAMA middleware architecture is described in section 3. A case study is described in section 4 which illustrates how context tracing is performed and how to build an application on top of the middleware. Section 5 concludes about the CAMA middleware and briefly describes our future work.

2. STATE OF THE ART

Our research lays at the intersection of two fields: context-aware pervasive computing and human-system interaction via multimodal dialogue. We present here three projects that concern these fields: Gaia, Conversational Interfaces and SmartKom.

Project Gaia [2] aims to construct an infrastructure for Smart Spaces, which are pervasive computing environments that encompass physical spaces. Applications in Gaia are developed using an agent-based middleware which uses ontologies to define the semantics of various contexts. The middleware doesn’t support specifically multimodal dialogue applications, because its ontologies don’t specify multimodal data. The middleware also doesn’t have the context tracing feature.

Project Conversational Interfaces [4] addresses the question of how to develop robust practical multimodal dialogue systems. It describes in detail an advanced discourse-processing structure for conversations between a user and a device. In this structure, the dialogue management part (which handles the user-system conversation) is clearly separated from the planning part (which is responsible for the implementation of tasks). This way increases the portability of the system components. The project prototype Witas [5] has context tracing. The system can answer what it is doing if the user raises the “why” questions. The limitation of this project is its prototype just allows the interaction between a user and only one device.

Project SmartKom [6] aims to develop a multimodal dialogue system that combines speech, gesture, and facial expressions for both user input and system output in pervasive environments. The SmartKom middleware has a distributed component architecture [7]. Ontology in SmartKom is used as a foundation for representing domain and application knowledge in order to enable natural conversation [8]. SmartKom uses a three-tiered representation of multimodal dialogue, consisting of a domain layer (which is responsible for planning), a discourse layer (which is responsible for dialogue handling) and a modality layer (which is responsible for capturing user input and presenting system output). SmartKom middleware doesn’t have the context tracing feature.

3. CAMA MIDDLEWARE ARCHITECTURE

3.1 Overview

The CAMA middleware aims at overcoming the main limitations of existing middlewares, by including the context tracing feature. The middleware architecture is agent-based as illustrated in Figure 1. It includes the following agents: the Ontology Server agent (OSa) to maintain an ontology, the Resource Manager agent (RMa) to manage the resources that the middleware can work with, the Situated agent (Sa) to capture situations which correspond to the high level contextual information, the So Manager agent (SMa) to coordinate all Situated agents, the Plan-based Process Manager agent (PPMa) to manage plan-based processes, the Context History Manager agent (CHMa) to maintain the context history and the Discourse Process Manager agent (DPMa) to manage discourse processes. These agents use two types of communication between each other: synchronous and asynchronous. For the first type, an agent sends a request to another and gets the response. For the second type, an agent subscribes for receiving events from another agent and events will be sent at any time. We describe all of these agents in detail in the following parts.

3.2 Ontology Server agent

The middleware requires only one Ontology Server agent. It maintains an ontology for describing the five models: the Resource Model (RM) specifies all kinds of resources that the middleware can work with, the Situation Model (SM) specifies all kinds of situations that the middleware can capture, the Conversational Data Model (CDM) specifies the structures of data to be exchanged between the agents, the
Plan Model (PM) specifies the plans that plan-based processes use, and the Context History Model (CHM) specifies how the information about an operation of a plan-based process is stored in the context history. We describe these five models in detail in part 3.8.

### 3.3 Resources and their manager

In Figure 1, a St denotes a resource interface which is responsible for event notifications and a Se denotes a resource interface for receiving requests to implement the services of the resource. The Resource Manager agent (RMa) obtains the Resource Model (RM) from the Ontology Server agent. When a resource joins the environment, it advertises its capability to other agents. The RMa then checks whether this information is described in the RM or not. If the information is described, the RMa registers this resource as part of the resources that the middleware can work with. Otherwise, this resource is ignored by the middleware. In Figure 1, the fact that the Se and St interfaces lay inside the RMa does not mean that they are part of it, but their resources are managed by this RMa. The middleware requires only one RMa.

### 3.4 Situated agents and their manager

A Situated agent, denoted by Sa in Figure 1, is responsible for capturing situations, which correspond to the high level contextual information. The Sa Manager agent (SaMa) obtains the Situation Model (SM) from the Ontology Server agent. When a resource joins the environment, it advertises its capability to other agents. The SaMa then checks whether this information is described in both the RM and the SM or not. If the information is described, the SaMa registers this resource as a Situated agent which belongs to the middleware. The middleware requires only one SaMa. Each Sa may:

- invoke the Resource Manager agent (RMa) for information about St interfaces, in order to subscribe for event notifications.
- invoke the Sa Manager agent (SaMa) for information about other Situated agents, in order to subscribe for event notifications.
- query for information from other agents.
- send events to other agents.

### 3.5 Plan-based processes and their manager

A plan-based process, denoted by Pb in Figure 1 executes a plan defined in the Plan Model. The Plan-based Process Manager agent (PPMa) is responsible for the instantiation of this process. Via its PPMa, a Pb can obtain information about situations from the Situated agents and can request the Se interfaces for services implementation. Also via its PPMa, a Pb can send the information about its operation to the Context History Manager agent for saving in a database.

### 3.6 Context History Manager agent

The Context History Manager agent (CHMa) receives the information about every operation of plan-based processes of the PPMa and stores it in a database. The Context History Model (CHM) regulates this storing, as described in part 3.8.6. This information is vital for context tracing. In part 3.9, we describe how context tracing can be performed using this information.

### 3.7 Discourse processes and their manager

A discourse process, denoted by Di in Figure 1, handles one conversation between a user and the system. It is managed by the Discourse Process Manager agent (DPMa). Via its DPMa, a Di can receive user inputs from the events sent by Situated agents, and can request for the operation of plan-based processes in the PPMa. A Di via its DPMa can request the CHM for the context history information. Discourse processes don’t belong to the middleware. A multimodal dialogue application includes one or several discourse processing modules built on top of the middleware. In the run time, the DPMa loads these modules to work as discourse processes.

### 3.8 Ontology in CAMA

#### 3.8.1 Why using ontology and how

Much research has shown the importance of using ontologies in the computing world [3] [9] [10]. In the context of the CAMA project, ontology is used to support:

- Semantic interoperability between entities: Interaction between entities (or resources) in the environment must base on common, well-defined concepts, so that there is no misunderstanding between them. For example, if an entity x wants to interact with an entity y, x needs to know which kind of interfaces y supports, and which protocols y accepts. The solution is that the middleware maintains somewhere, for example, in one of its agent, an ontology about all kinds of entities it needs to utilize. Entity x or y just consults this agent in order to know about each other. When a new entity enters the environment, the application decides whether this is the one it can utilize or not, based on the ontology.
• Context-awareness: Various types of contextual information that can be used by the middleware must be well-defined so that different entities have a common understanding of context. Also, there needs to have mechanisms to specify plans and how these plans work in different contexts. These mechanisms need to be based on a common well-defined structure.

The CAMA middleware ontology (also named CAMA ontology) is specified using the Web Ontology Language (OWL) [11]. OWL allows designers to describe the structure of a domain in terms of classes and properties. The CAMA ontology includes the top-level ontology and the five models as described in the following parts.

3.8.2 Top-level ontology

In the CAMA ontology, every class is a sub-class of Entity. Class Person represents a person in general. Class Device represents a resource like a light, a television, a software agent that provides services, or a Situated agent. Class Space represents a space like building, house, or room. Class Document represents a paper, book, or email. Class Situation represents a situation. Class Plan represents a plan. Class Co-Data represents the structure of data to be exchanged between agents in the application. Class Process represents the information about the operation progress of a plan-based process. We explain these classes in more detail when describing the five models in the next parts.

3.8.3 Resource Model (RM)

This model contains the instances of class Device. Each instance specifies the information about a resource, or a type of resources that the middleware can work with. A resource can be a physical device, a software agent that provides services, or a Situated agent. The structure of each instance is defined via its class properties. A property is either required or optional to appear in an instance. These properties are:

• ID (required): The resource identifier.
• ResourceType (required): A physical device, a software agent that provides services, or a Situated agent.
• StateList (optional): It contains one or more instances of class StateVariable. Each instance of this class is the declaration of a variable with a certain type. In the run time, if the value stored in this variable changes, an event is sent out with the new changed value. In case the resource is a Situated agent, the type of this variable is the name of a situation class (see 3.8.4) which represents the situations that the Situated agent capture (e.g., class Location, in part 3.8.4) and the allowed values of this variable contains the instances of this situation class.
• ServiceList (optional): It contains one or more instances of class Service. Each instance specifies a service that the resource provides.

3.8.4 Situation Model (SM)

This model describes all situations, which correspond to the high level contextual information. A situation is represented as a predicate where its name is the type of the situation that is being described, e.g., position, time. By doing this, different kinds of situations have a simple and uniform representation. Example of some situations are:

• Location (Peter, Beside, PC, 001.012)
• PrinterStatus (diuf1 printer queue, is, empty)
• Temperature (room 121, “==”, 30)

Each type of situations corresponds to a class -- let’s call situation class - which is a sub-class of class Situation. For example, class Location represents all kinds of situations concerning the position of a person. The situation “Peter is sitting at a PC” is represented in the model as:

```xml
<Location>
  <PersonName>Peter</PersonName>
  <Position>Beside</Position>
  <ObjectID>001.012</ObjectID>
</Location>
```

3.8.5 Plan Model (PM)

This model contains the instances of class Plan. Each instance specifies a plan. The class properties specify an instance structure. A property is either required or optional to appear in an instance. These properties are:

• ID (required): The plan identifier.
• Acti-Condition (required): This specifies a situation, which is called the activating condition of the plan. If this situation holds, we say this plan is activated and the plan-based process (which uses this plan) starts its operation (we explain this operation when we describe the Operation property).
• Text (optional): The explanation about the plan in natural language.
• Properties (required): They specify input parameters and states of the process using this plan.
• Goal (required): This specifies a situation. If it holds then the operation based on this plan ends.
• Exception (required): This also describes a situation. If it holds then the operation based on this plan ends.
• Operation (required): This is the schedule for the operation as soon as the Acti-Condition holds. It includes a number of instances of class Task. Let’s call them t-instances. Each t-instance is a sequence of p-instances, which are either the instances of other plans or the calls to some services. An operation based on this schedule is performed as follows. For every t-instance, if the first p-instance is an instance of another plan then as soon as the Acti-Condition specified in this p-instance holds, a new plan-based process is instantiated by the PPMa. This process’s operation ends if the p-instance’s goal is achieved or the p-instance’s exception occurs. If the first p-instance is a call to a service then this service is implemented. Next, the second p-instance of this t-instance is treated as the first was, and so on. The whole operation ends in case either the plan’s goal is achieved, or the exception occurs.
• ProcessEnd (optional): This specifies a situation. If this situation holds then the process using this plan is removed from the Plan-based Process Manager agent.
properties are:

- ID (required): The instance identifier
- Based-plan (required): The ID of the plan that the plan-based process uses.
- Start-up (required): Information about the starting time (which is an instance of class FromTime) of the operation, also the input values (which is an instance of class InputPara).
- End (optional): Information about the ending time (which is an instance of class ToTime) of the operation, also values that make the plan’s goal achieved (which is an instance of class F-Goal), or the exception occurs (which is an instance of class F-Exception). This property will not appear if the operation of the plan-based process is not ended yet.
- Operation-R: It includes a number of instances of class Task-R. Each instance is a sequence of instances of class Process or class Service.

Figure 2 presents an example of an instance of class Process. It is the operation result of a plan-based process using the plan of Figure 2. This is the case when the operation is successful - the goal is met. The operation of five plan’s instances a, b, c, d, e in Figure 2 results in five instances pa, pb, pc, pd and pe of class Process.

3.8.7 Conversational Data Model (CDM)

Agents need to know well-defined structures of data to be exchanged between them. These structures are represented by the conversational data classes, which are sub-classes of class Co-Data. The Conversational Data Model (CDM) organizes these classes hierarchically. Figure 4 represents part of the hierarchy of the conversational data classes. Class AG-OS represents the data to be exchanged between the Ontology Server agent and other agents. Class R-PP represents the data to be exchanged between a resource and the Plan-based Process Manager agent. Class R-DP represents the data to be exchanged between a resource and the Discourse Process Manager agent. Class R-CH represents the data to be exchanged between a resource and the Context History Manager agent. Class DP-PP represents the data to be exchanged between the Discourse Process Manager agent and the Plan-based Process Manager agent. Class PP-CH represents the data to be exchanged between the Plan-based Process Manager agent and the Context History Manager agent. As shown in Figure 4, there are three classes (GiveInfor, Command, Question) which represent three types of data to be exchanged between a resource and the Discourse Process Manager agent: information provided (e.g., “The coffee is ready”), command (e.g., “Switch this light on please”), and question (e.g., “Why do you show this message here?”). An instance of class ObjectCommandOn captures the user information in case the user says “Switch it on please” while pointing his finger to a light. The properties of this class are:

- DeviceType: The type of the device, for example: Light.
- DeviceID: The device identifier.

3.9 Context tracing support

The database maintained by the Context History Manager agent (CHMa) supports context tracing. For example, let’s go back to Figure 3. Assume the user asks the system about...
the existence of a current situation, which is described in the instance \textit{pc} of class \textit{Process}. The system can tell the user about:

- Which plan has been used in order to produce \textit{pc}
- What are the operations that happened before producing \textit{pc} (these operations produce \textit{pa}, then \textit{pb}).
- The whole operation of the process that produces instance \textit{pp}.

The case study in section 4 give some illustrations of context tracing.

4. CASE STUDY

We have developed the CAMA middleware and a prototype application named OFFICE. This application allows users to interact with devices in an office environment, using multimodal dialogues. This application contains several discourse processing modules built on top of the CAMA middleware. First, we present an user - system interaction scenario that OFFICE covers. After that, we describe the design and implementation of the OFFICE discourse processing modules which serve for that scenario.

4.1 Scenario

Assume a person is in her working room. She is able to interact with devices, using speech and gesture. A dialogue example is shown in Figure 5. It illustrates the possibility of the system to understand the user command, then to act according to the context, and to explain how this acting has happened.

From line 1 to line 4, the user requires the system to implement a plan for “today and tomorrow”. Assume that one day later, the user forgets about that requirement, so that the appearance of a message on the PC screen makes her wondering about the reason for that. She requires the system to explain about that, as in line 5. Using context tracing, the system knows that it has applied plan \textit{P002} to adapt to the user upon the context: “you are at the computer”. This plan is part of another plan \textit{P001}. Consequently, it answers to the user, as shown in line 6 and line 8.

![Figure 4: Part of class hierarchy of the conversational data model](image)

![Figure 5: A scenario example](image)

1. User (U): Inform me when a mail from Michele arrives!
2. System (S): You want me to do it for today only, or for some days?
4. S: OK.
5. U: Why do you show this message here?
6. S: Because I apply plan P002, saying: If there is a message for you when you are at the computer then display it on the screen.
7. U: Why do you apply this plan?
8. S: Because this is a part of plan P001, saying: If a mail from Michele arrives then inform you.

4.2 Design and implementation

The CAMA middleware agents (OS\textsubscript{a}, RM\textsubscript{a}, Sa\textsubscript{Ma}, Sa, PP\textsubscript{Ma}, CH\textsubscript{Ma}, DP\textsubscript{Ma}) and the physical devices, software agents that the middleware uses communicate with each other using the UPnP protocol [12] [13]. To serve for the scenario in Figure 5 only, the following physical devices and agents are used by the middleware. The physical devices include a PC and a sensor set on the chair near by the PC. The agents includes an OS\textsubscript{a}, RM\textsubscript{a}, Sa\textsubscript{Ma}, PP\textsubscript{Ma}, CH\textsubscript{Ma}, DP\textsubscript{Ma} and:

- \textit{Speech recognizer and natural language understanding} (SR-NLU): This agent (which connects to a microphone) captures the user speech and processes it as following. The NUANCE speech recognizer [14] (which is embedded inside this agent) produces an utterance from the speech. To parse the utterance, the information state approach is used [15]. The NUANCE grammar for the parsing is generated by the REGULUS tool [16] [17]. By using the NUANCE natural language understanding module (which is embedded inside the agent), the utterance is translated in to a dialogue move (DMOVE), which is a logic form in the information state approach.

- \textit{Pointing Object} (PoOb): This agent captures information about the object that user is pointing to. Because the vision techniques for capturing object information are not our research, we decide to simulate this kind of action “pointing to an object” by a mouse click on a window that contains the object information.

- \textit{User Input Channel} (UIC): This Situated agent receives events from the SR-NLU and the PoOb. It captures the situation likes a user is speaking a sentence while pointing his finger to an object (e.g., line 5 in Figure 5).

- \textit{MsgWindow}: This is a window on the PC screen. A message sent by the system to the user is displayed there.

- \textit{Speaker}: This agent connects to a loudspeaker. If a message is sent to this agent, it uses the NUANCE...
This paper describes a context-aware middleware for multimodal dialogue applications having the context tracing feature. The middleware is agent-based and an ontology is used to describe various concepts such as resources, situations, plans, structure of context history, and data to be exchanged between the agents. This middleware has been validated through a case study. This case study has illustrated how context tracing is performed. It has also shown that the development of an application on top of the middleware is straightforward and simple.

Context tracing starts from a considering situation. The question is how the middleware can identify this situation based on some given attributes. This possibility of identification is called context searching, as we mention in section 1. For example, the middleware can identify what moment during yesterday the printer was blocked. In the near future, we intend to include the context searching feature in the CAMA middleware.

6. REFERENCES

Communication and Information Technology, St. Thomas, Virgin Island, USA, November 2002.


