Xipho: An Agent-Oriented Methodology for Context-Aware Application Development

1 Introduction

Humans have an inherent understanding of the context in which they act and interact. Increasingly, applications also strive to adapt to the contexts of their human users, especially applications for smart devices. Such *context-aware applications* are increasingly popular in domains such as personalization, information retrieval, privacy management, entertainment, automatic task execution, and so on.

We understand context as a cognitive notion. A context-aware application must, somehow, capture its users' mental models of context—specific to the scope of the specific application or application domain. Thus the engineering of a context-aware application is a nontrivial challenge. We introduce Xipho, a novel methodology to assist an application developer in systematically eliciting the intended users' mental models of context by relating context to other cognitive notions such as users' goals, plans, and dependencies.

2 Background: Tropos

Xipho extends the Tropos methodology [1]. Tropos describes the generic steps of agent-oriented software development spanning *requirements* acquisition, *design*, and *implementation*. Tropos employs model-driven development. The crux of Tropos is to model the *system-to-be* from a *system-as-is* model, and to iteratively refine the system-to-be model to derive a detailed specification of the system-to-be. The detailed specification can then be mapped to an implementation.

A Tropos model includes the following constructs (which form the Tropos metamodel).

Actor, a social, physical, or software agent (or a *role* of an agent). An actor has goals within a system.

Goal, a strategic interest of an actor. A *hard goal* has a crisp satisfactory condition. A *soft goal* has no clear-cut definition or criteria for deciding whether it is satisfied or not. Thus hard goals are *satisfied* whereas soft goals are *satisficed*.

Plan, an abstraction of doing something. Executing a plan is a means of satisfying or satisficing a goal.

Resource, a physical or information entity.

Dependency, a relationship between two actors—a *depender* and a *dependee*—indicating that the depender depends on the dependee for accomplishing a goal, executing a plan, or furnishing a resource. The object on which a dependency arises is a *dependum* (goal, plan, or resource).

Belief, an actor's knowledge of the world.

Capability, an actor's ability to choose and execute a plan to fulfill a goal, given certain beliefs and resources.

Figures 1 and 2 show a graphical notation used to represent Tropos models.

3 The Xipho Methodology

Xipho operates under the broader rubric of Tropos. However, Xipho addresses challenges specific to context-aware application development which Tropos does not address. The objectives of Xipho are to assist a developer in (i) acquiring and analyzing contextual requirements, (ii) designing the application and deriving a specification, and (iii) implementing the application. Throughout the development process, Xipho emphasizes the need to systematically capture the *why*, besides the *what* and the *how*, of the application being developed.

Table 1 provides an overview of Xipho. Xipho begins with the actor modeling activity (adopted from Tropos). Steps 2–5 are Xipho's extensions to Tropos. Step 2 describes a technique to analyze the requirements and capture contextual beliefs and resources that influence the application. The design steps (3 and 4) describe techniques to derive an application specification consisting of a context information model and the contextual capabilities of the application. Finally, Step 5 describes the implementation and an approach to reuse a middleware component to simplify the implementation.

Table 1: An overview of the Xipho methodology. Step 1 is described by Tropos. Steps 2–5 are Xipho's extensions to Tropos.

#	**Step**	**Activity**
1	Actor modeling (Tropos)	Model the intended setting in which the application is used
2	Context-means analysis	Identify contextual beliefs and resources
3	Context information modeling	Identify context abstractions and levels
4	Contextual capability modeling	Map context levels to capabilities
		Specify the application as a set of contextual capabilities
5	Implementation	Implement contextual capabilities

Next, we describe a case study to serve as a running example. The case study involves the engineering of a context-aware application. We demonstrate Xipho by applying it to develop the context-aware application step by step.

3.1 A Case Study: Ringer Manager Application

The application to be developed in the case study is a *Ringer Manager Application (RMA)*. The RMA is a smart phone application whose usage scenario would be familiar to most cell phone users. The RMA helps a user better handle incoming calls on his or her cell phone.

- First, the RMA sets an appropriate ringer mode on the user's cell phone based on the user's context at the time of an incoming call. The alternatives are to set the ringer mode to be *silent*, *vibrate*, or *loud* (we assume these modes to be mutually exclusive).
- Second, if the user doesn't answer the call, the RMA sends a notification to the caller. The notification can be a generic message (e.g., "leave your name and number"), or a message containing information about the user's context at the time of missing the call. The context information can be abstract (e.g., "in a meeting") or detailed (e.g., "in a meeting with Bob at the Starbucks from 10 am to 12 pm").

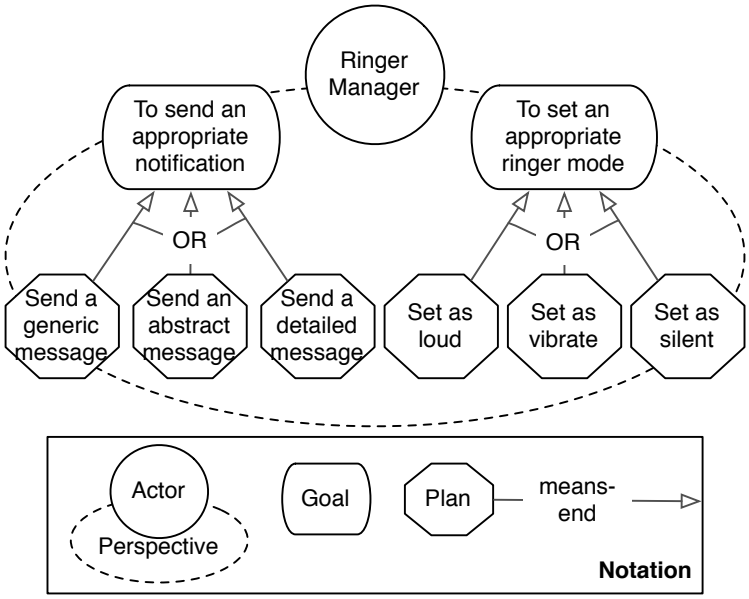

Figure 1: An initial model of the RMA.

Figure 1 shows an initial model of the RMA capturing the above description as goals and plans. The model is constructed without applying Tropos or Xipho (and it is meant to represent a model one might construct in an ad hoc manner). The initial model, although simple, suffers from three major limitations. (1) The model captures *what* the RMA does, and with missing details, *how*. But, clearly there is no trace of not *why*. (2) The model does not capture how users' contexts influence the decisions RMA makes. (3) Finally, the model is not exhaustive. These limitations make the implementation of RMA nontrivial. The following steps describe techniques to iteratively refine the initial RMA model.

3.2 Step 1: Actor Modeling

Xipho's first step is to model the intended setting in which an application is used. The objective of this step is to explicitly capture what the application does and more importantly, why it does so. We employ the actor modeling activity described by Tropos for this purpose. Actor modeling consists of substeps to identify:

Actors involved in the setting in which the application is employed. These are the primary users of the applications as well as those indirectly influenced by the application.

Goals and plans of each actor within the setting.

Dependencies among actors to accomplish goals or execute plans.

Application as an actor itself and update the goals, plans, and dependencies.

We imagine the intended setting of the case study to be as follows. Each episode in the setting starts when a *caller* tries to reach a *callee*. The callee wants to be reachable unless he wants to work uninterrupted. The callee's plan is to answer the call if he wants to be reachable and not answer otherwise. The callee might decide to answer or not depending on (i) whether he disturbs a *neighbor* by answering or (ii) if the caller has a pressing need to reach him. In such cases, the callee depends on the neighbor or the caller to provide appropriate information. The callee sets an appropriate ringer mode on his cell phone to help him answer or not (e.g., loud or vibrate to answer, silent to not). Next, when the callee doesn't answer a call, he might want to notify the caller of a reason (e.g., busy, in a class, talking to boss, and so on) or ignore depending on who the caller is. If he decides to notify the caller, he wants to preserve privacy by disclosing only the necessary details. The actor model in Figure 2 captures this setting.

However, the above setting is quite inefficient. First, it relies on the callee to manually set an appropriate ringer mode and send an appropriate notification. From the caller perspective, there is no effective way of expressing a pressing need to reach the callee (calling repeatedly doesn't help if the phone is silent). From a neighbor's perspective, it is tedious to let each callee know the neighbor's intent to be not disturbed.

The RMA intends to make the call handling more efficient. For simplicity, we explore the RMA only from the perspective of the callee, the primary user of the application. We assume the secondary users—caller and neighbor—to have appropriate mechanisms to provide the information required by the RMA (possibly through another application). The actor model in Figure 3 introduces RMA, the system-to-be actor, and expands its perspective. The RMA acts on behalf of the callee and accordingly adopts the callee's goals.

In contrast to the initial RMA model in Figure 1, the RMA model in Figure 3 is already an improvement. Whereas the initial model captured what the RMA did (e.g., *to set an appropriate ringer mode*), the improved model captures why it is doing so by linking the RMA's plans to its users' goals (e.g, *set as silent* to let the user accomplish the goal *to be uninterrupted*).

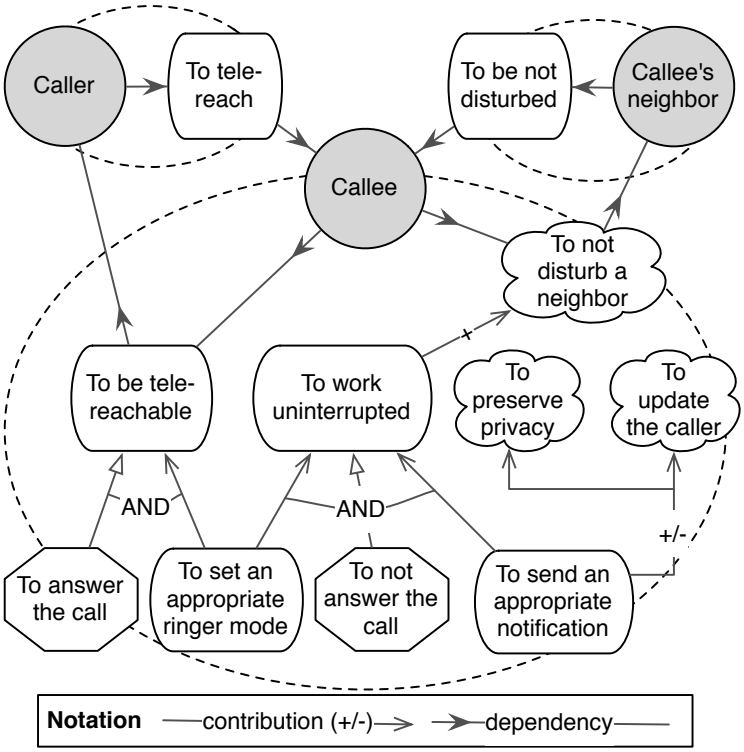

Figure 2: An actor model with the callee's perspective expanded. The actors are highlighted.

3.3 Step 2: Context-Means Analysis

A context-aware application needs to make decisions based on the contexts of one or more of the actors involved. The objective of *context-means analysis* is to identify scenarios where an actor's context provides the application a means of making a decision. We describe four such scenarios and provide an example for each scenario from the case study.

OR-decomposition, where a subset of the decomposed goals or plans are to be chosen over the rest based on context. For example, whether the ringer mode should be *set as silent* or *set as vibrate* could depend on context—vibrate might not interrupt in a *seminar*, but it might in a *one-to-one meeting*.

Conflicting goals, where the goal chosen for accomplishment depends on context. For example, the RMA must choose between the callee's goals *to be tele-reachable* and *to work uninterrupted*. This decision, for instance, could be based on the callee's context—weather *busy* or not.

Soft goal, where the extent to which to satisfice the goal depends on context. For example, each ringer mode satisfices the soft goal *to not disturb a neighbor* to a different extent. The chosen ringer mode could be based on who the neighbor is, e.g., a *set as silent* near a *colleague*, *set as vibrate* near a *family member*, and *set as loud* near a *stranger*.

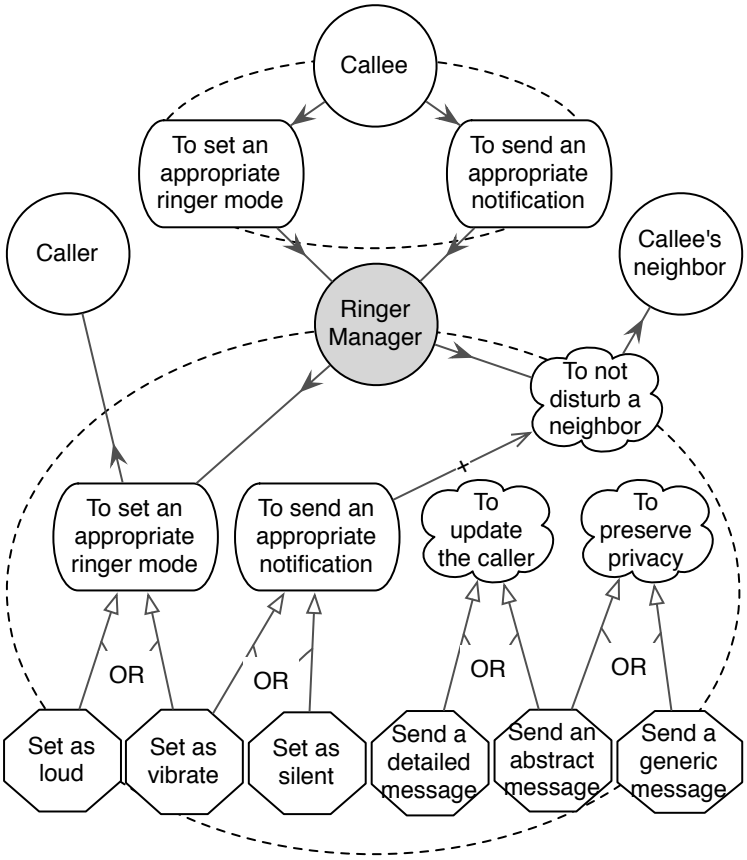

Figure 3: An actor model expanding the RMA's perspective. The RMA acts on behalf of the callee and adopts his goals.

Dependency, where the dependum can be refined based on context. For example, the dependum *to be tele-reachable* can be refined to context because the actual dependency is that the callee depends on the caller to provide a context resource (e.g., caller's context indicating if he has a pressing need to reach the callee).

A developer must identify all scenarios of each type described above. Further, he must perform one of the following substeps depending on the scenario.

- If the scenario is influenced by the context of the primary user of the application (the *callee* in the case study), capture the influence as a belief and add context-means links from the belief to each construct involved in the scenario.
- If the scenario is influenced by the context of a secondary user (the *caller* and *neighbor* in the case study), capture the context as a resource and add a dependency (or update existing dependency) to have the context resource as the dependum. Also, add context-means links from the resource to the constructs involved in the scenario.

Figure 4 shows the result of the context-means analysis of the RMA.

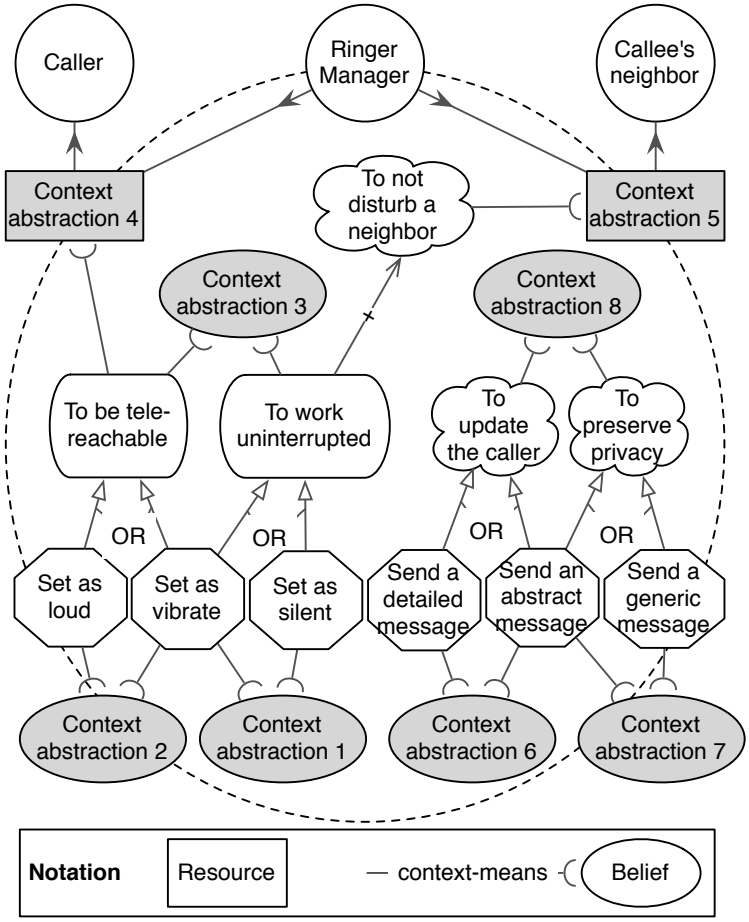

Figure 4: The RMA's actor model after the context-means analysis.

3.4 Step 3: Context Information Modeling

Each time there was notion of context in the previous step, we used a "context abstraction" as a place holder. However, what are these abstractions and how to derive them systematically? The objective of Step 3 is to answer this question.

The notion on context has been defined by several authors [2]. At an abstract level context is defined as "any information relevant to an interaction between a user and an application" [3]. Although generality is desired for a definition, the current purpose demands specific details for two reasons. (1) A motivation for model-driven development is to document the thought process of a developer—the more details a model can capture, the better documented the developer's intuitions are. (2) In the next phase, the application model is used to derive a system specification. A detailed model leads to a detailed specification and potentially, makes the application implementation easier.

In order to derive a context information model, a developer must identify:

Meaningful context abstractions for each generic abstraction found in the previous step.

The context-based scenario influenced by the generic abstraction provides a guideline

for scoping down the abstraction. In general, the new set of abstractions must be specific to the scenario, yet generic enough to cover all the situations associated with the scenario. For example, in Figure 4, the *Context abstraction 2* is used to decide to set the ringer mode as *loud* or *vibrate* In this case, the context abstraction can be scoped down to a spatial abstraction of *proximity to phone*, because all that matters to decide the the ringer mode to be *loud* or *vibrate* is the user's proximity to the phone (note that the decision to not set as *silent* must have been already made, using *Context abstraction 3*).

Context levels of an abstraction, where appropriate. A context level of an abstraction is the set of all situations that must be treated ordinarily within the scope of the abstraction. For example, the context levels of the abstraction *Proximity to phone* could be *Near* and *Far*. However, it might not be possible or desirable to identify the context levels for each abstraction during development. It might be desirable to elicit the context levels of an abstraction at run time. We refer to such an abstraction as an *open abstraction*. The intuition is that the levels of an open abstractions are to be subjective and restricting them at development time reduces the generality of the application. For example, the callee's *activity* is an open abstraction. It is desirable to elicit from the user, for example, which of his activities are to be uninterrupted.

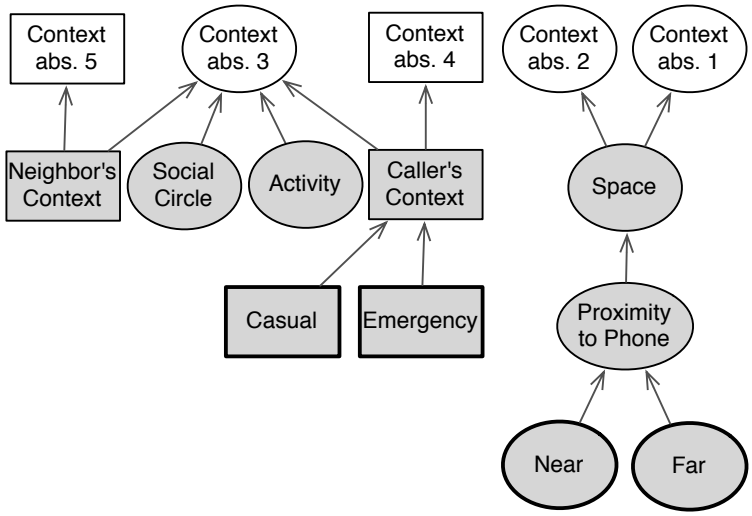

Figure 5: A context information model for the RMA. Each context level is highlighted with a thick border.

In this and the next steps, we restrict the case study to the ringer management aspect of the RMA (and omit the notification aspect), without loss of generality. Figure 5 shows a context information model derived for the RMA. We employed the context abstractions of *space*, *activity*, and *social circle* in an existing metamodel [4]. The choice of a metamodel is loosely coupled with Xipho, though.

3.5 Step 4: Contextual Capability Modeling

The objective of this step if to obtain an application specification. A detailed specification could simplify the implementation phase. Xipho's specification of a context-aware application is a set of *contextual capabilities*. A contextual capability of an application is a capability conditioned on a context abstraction being at a context level. A contextual capability can be represented as a rule, e.g., $\{Proximity\ to\ phone = Far\} \rightarrow \{Set\ as\ loud\}$.

In order to derive an application specification, a developer must perform the following substeps.

- Identify the capabilities of the application. Essentially, the capabilities are the plans an application can execute. Note, however, that additional plans can be introduced at this stage to represent the choices the application could make (e.g., such as to choose a goal to accomplish from a set of conflicting goals).
- Identify the context abstraction on which each capability found above is to be conditioned. That is, replace the generic context abstraction place holders added in Step 2 with specific abstractions found in Step 3.

Table 2: A specification of the RMA as a set of contextual capabilities.

Contextual Capability = {**Contextual Condition**}		→ {**Capability**}
C_1 =	$\{Activity = A_1 \wedge Social\ circle = S_1 \wedge Neighbor's\ context = N_1 \wedge Caller's\ context = Emergency\}$	$\rightarrow \{Set\ as\ loud \vee Set\ as\ vibrate\}$
C_2 =	$\{Activity = A_1 \wedge Social\ circle = S_1 \wedge Neighbor's\ context = N_1 \wedge Caller's\ context = Casual\}$	$\rightarrow \{Set\ as\ loud \vee Set\ as\ vibrate\}$
C_3 =	$\{Activity = A_2 \wedge Social\ circle = S_2 \wedge Neighbor's\ context = N_2 \wedge Caller's\ context = Emergency\}$	$\rightarrow \{Set\ as\ loud \vee Set\ as\ vibrate\}$
C_4 =	$\{Activity = A_2 \wedge Social\ circle = S_2 \wedge Neighbor's\ context = N_2 \wedge Caller's\ context = Casual\}$	$\rightarrow \{Set\ as\ silent \vee Set\ as\ vibrate\}$
C_5 =	$\{(C_1 \vee C_2) \wedge Proximity\ to\ phone = Near\}$	$\rightarrow \{Set\ as\ vibrate\}$
C_6 =	$\{(C_1 \vee C_2) \wedge Proximity\ to\ phone = Far\}$	$\rightarrow \{Set\ as\ loud\}$
C_7 =	$\{(C_3 \vee C_4) \wedge Proximity\ to\ phone = Near\}$	$\rightarrow \{Set\ as\ silent\}$
C_8 =	$\{(C_3 \vee C_4) \wedge Proximity\ to\ phone = Far\}$	$\rightarrow \{Set\ as\ vibrate\}$

Figure 6 shows a refined actor model of the RMA that combines context abstractions and capabilities. This model can be used to generate a detailed set of contextual capabilities. Table 2 lists the contextual capabilities we identified for the RMA. Note that we use variables to capture the context levels of an open abstraction (the number of variables required equals the number of alternatives available in the scenario influenced by the abstraction).

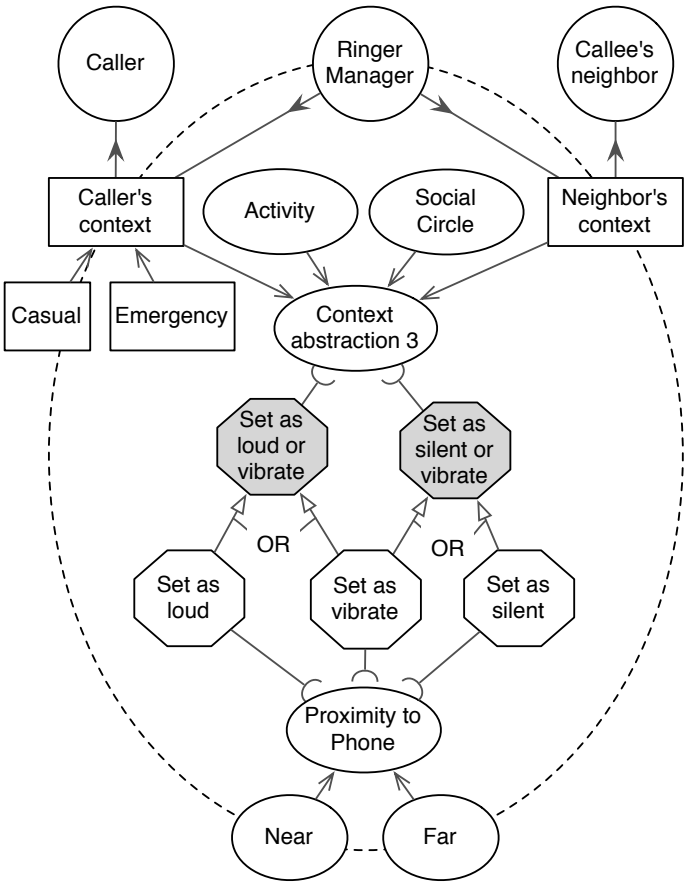

Figure 6: A refined actor model of the RMA combining the context information model and capabilities. The new capabilities introduced are highlighted.

3.6 Step 5: Implementation

The objective of this step is to implement the application. In this step, the tasks for a developer are to (i) implement each capability, e.g., set the ringer mode to be *silent*, and (ii) contextualize each capability, i.e., implement the rules derived in the previous step. However, there are two prerequisites for an application to exercise contextual capabilities—the application must be equipped with:

Techniques to elicit context levels for each open context abstraction from end-users. As the levels of each abstraction are elicited at run time, the set of contextual capabilities can be expanded within the application. For example, the RMA may elicit from a user that the required levels for the abstraction *Activity* are *Home*, *Office*, and *Elsewhere*.

Techniques to reason about each context level of a context abstraction. For an application to exercise a contextual capability, it must know when the antecedent of the

contextual capability is true. Thus, the application must have a mechanism to determine if an abstraction is at a specific level, when required, e.g., to determine if the user's *Activity* at the time of receiving a call is *Office*.

References

[1] Paolo Bresciani, Anna Perini, Paolo Giorgini, Fausto Giunchiglia, and John Mylopoulos. Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, May 2004.

[2] Matthias Baldauf, Schahram Dustdar, and Florian Rosenberg. A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 2(4):263–277, June 2007.

[3] Anind K. Dey, Gregory D. Abowd, and Daniel Salber. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction*, 16(2):97–166, December 2001.

[4] Pradeep K. Murukannaiah and Munindar P. Singh. Platys: An empirically evaluated middleware for place-aware application development. http://www.csc.ncsu.edu/faculty/mpsingh/papers/tmp/Platys-developer.pdf, 2012.