# A Context-Aware Application Development Methodology

We summarize Tropos [1], a model-driven software development methodology. First, we describe the methodology briefly. Next, we demonstrate the methodology in a case study.

## 1 A Summary of Tropos

Tropos spans the following five phases of software development.

**Early requirements** phase involves identifying the domain stakeholders and dependencies among them. This is also called as modeling the *system-as-is*.

**Late requirements** introduces an additional actor to the model called as *system-to-be*. The system-to-be represents the changes to the system-as-is that the development activity wants to bring about.

**Architectural design** involves mapping the system-to-be actor into one or more agents and identifying dependencies among agents at the level of the data and control structures.

**Detailed design** involves specifying capabilities each agent needs to accomplish its plans or to execute its plans.

**Implementation** involves mapping the agent specification into software on the desired platform.

In each phase a system model is constructed. A Tropos model consists of the concepts and relationships described in the Tropos metamodel summarized in Table 1. Tropos describes five modeling activities to perform in one or more of the development phases. Each modeling activity produces a model which is a refinement over the previous model. We describe the modeling activities next.

**Actor modeling** involves identifying the actors, and their goals and plans. In the early requirements stage, the actors are the domain stakeholders (and the existing software actors acting on behalf of the stakeholders), whereas in the late requirements phase an additional actor known as the system-to-be actor is introduced.

**Dependency modeling** involves identifying the dependencies among the actors centered around goals, plans, or resources. New dependencies can be elicited during each development phase as the models (as-is or to-be) are refined. While a dependency captures the dependum, it doesn't capture how the dependee brings (or fails to bring) about the dependum.

Table 1: Concepts and relationships described in the Tropos metamodel.

| **Construct** | **Description** |
|---|---|
| Actor | An entity that has goals and intentionality with in a system. An actor can be a physical, social, or software agent as well as a *role* of an agent. |
| Goal | A strategic interest of an actor. A goal can be a hard goal or soft goal. A soft goal, unlike a hard goal, has no clear cut-definition or criteria for deciding whether it is satisfied or not. The distinction is captured by the terminology that 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 informational entity. |
| Dependency | A relationship between two actors—a *depender* and a *dependee*—indicating that the depender depends on the dependee for accomplishing a goal or executing a plan, or to furnish a resource. The object around which a dependency centers is called a *dependum* (goal, plan, or resource). |
| Belief | An actor's knowledge of the world. |
| Capability | An actor's ability to define, and choose and execute a plan to fulfill a goal, given certain world conditions and in the presence of specific events. |

**Goal modeling** involves eliciting a finer structure to goals. The AND/OR decomposition breaks down a goal logically. The *means-end analysis* involves identifying the plans as a means of accomplishing goals. The *contribution analysis* involves identifying goals that contribute to other goals.

**Plan modeling** is akin to goal modeling, but applied to plans.

**Capability modeling** is performed after the architectural design. It involves identifying the capabilities required by the system-to-be subactors. A capability may be an *individual* capability required to fulfill a goal or to execute a plan, or a *social* capability required to handle dependencies.

## 2 Case Study: Cellphone Call Handling

We describe a scenario and use it as a case study to demonstrate PlatysM. The scenario is that of cellphone call handling. The scenario is realistic and one that is commonly encountered by all cell phone users. However, as we describe, it is also a nontrivial case for study.

Each episode in the scenario 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. Further, the callee might decide to answer or not depending on whether 1) he disturbs a *colleague* by answering or 2) the caller has a pressing need to reach. In such cases, the callee depends on the colleague or the caller to provide appropriate information. Further, the callee relies on setting an appropriate ringer mode on his cellphone for either purpose (e.g., loud to answer and 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, boss around, 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. Figure 1 shows a system-as-is model of the call handling scenario.

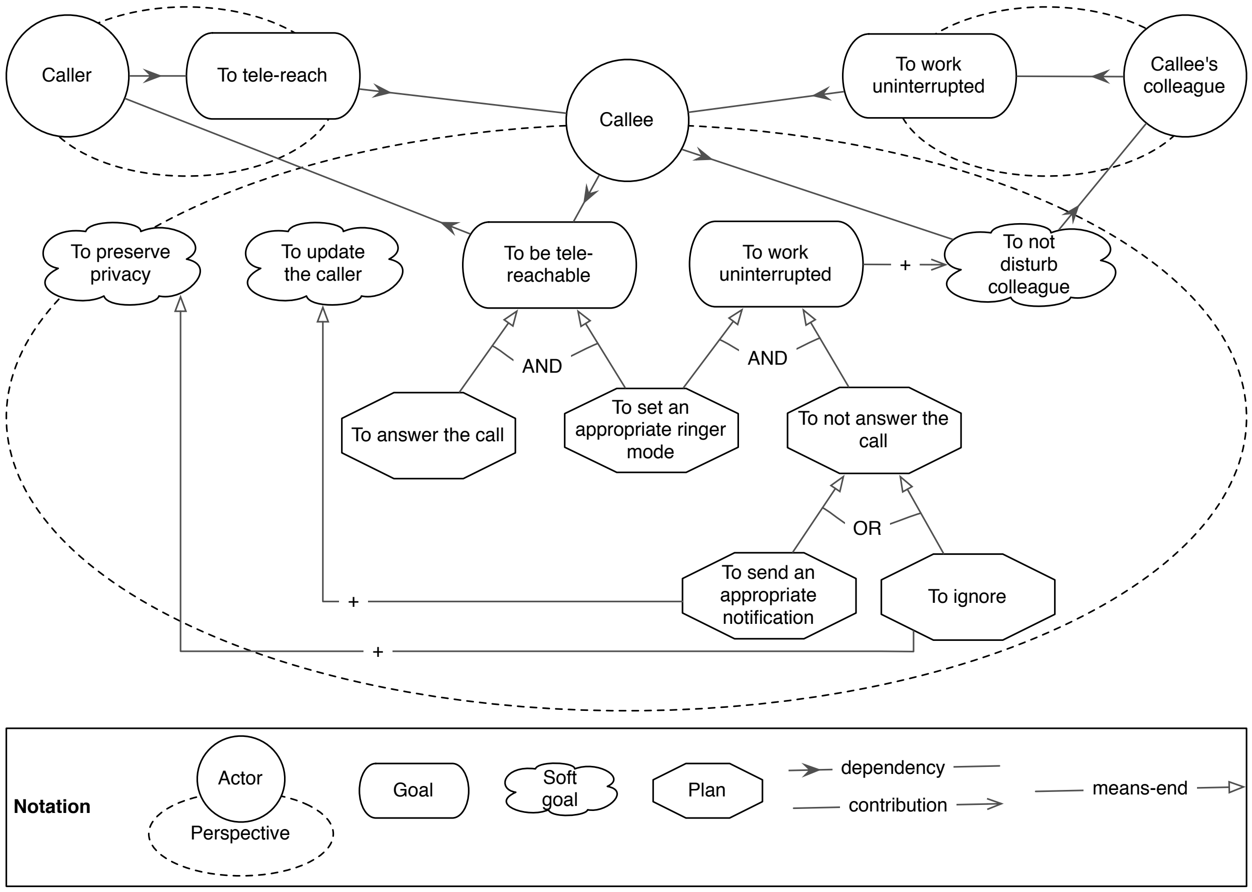

Figure 1: A system-as-is model of the call handling scenario. Only the callee's perspective is elaborated.

Although it may seem trite, the calling handling system-as-is is quite inefficient. From the callee's perspective, it relies on him 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 (calling repeatedly doesn't help if the phone is silent). From a colleague's perspective, it is tedious to let each callee know the colleague's intent to work uninterrupted. The system-to-be intends to make the call handling more efficient. For simplicity, we explore the system-to-be from the callee's usage perspective only. As shown in Figure 2, the system-to-be intends to automate two tasks the callee's performs manually in the system-as-is. The tasks are modeled as soft goals because there are no clear-cut criteria, at this stage, for the system-to-be to to decide if it satisfies the callee.

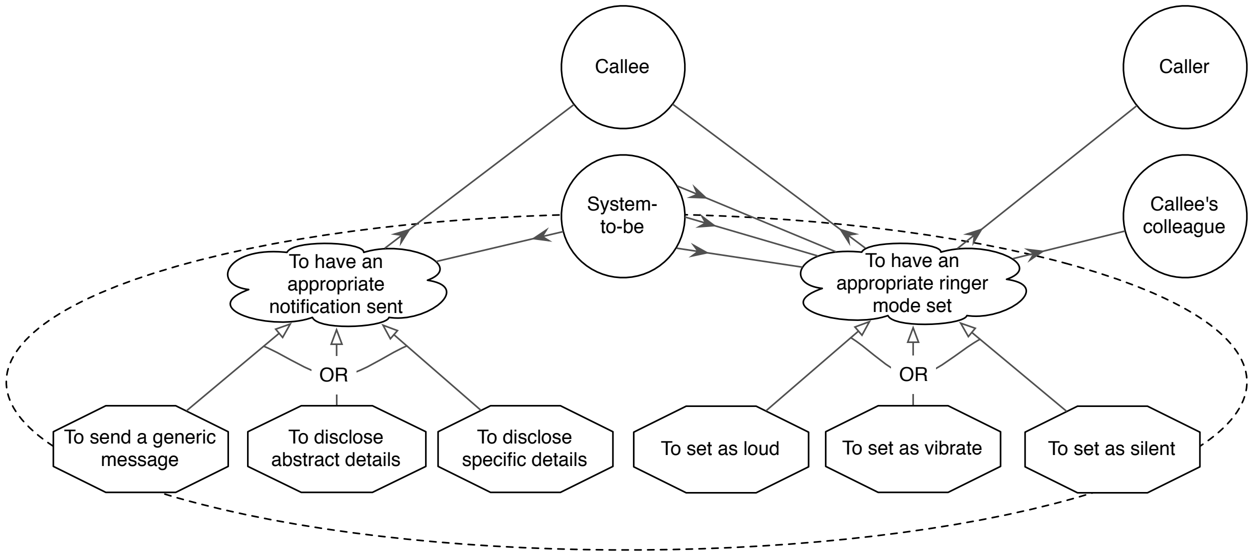
Callee
Caller
System-to-be
Callee's colleague
To have an appropriate notification sent
To have an appropriate ringer mode set
OR
OR
To send a generic message
To disclose abstract details
To disclose specific details
To set as loud
To set as vibrate
To set as silent

Figure 2: A system-to-be model of the call handling scenario. The system-to-be automates two tasks the callee performed manually in the system-as-is.

## 3 PlatysM Methodology

The system models (both as-is and to-be) elicited in the previous section capture the actors, their goals and plans, and dependencies between actors at the level of goals and plans. However, they fail to capture the role of context within the system. While the need to capture context might vary from system to system, for a system such as the one in our case study context is mainstream. As we show in this section, context can contribute to the major chunk of requirements and specification of a system. Failing to capture the role of context in such a system makes developers' intuition about actors' context to get buried in the implementation. This is precisely the problem Tropos was set to solve.

We recognize that capturing the role of context within a system is nontrivial. With that intuition, we provide an end-to-end methodology for systematically capturing the role of context with in a system. The methodology adds two steps (steps 1 and 2) to the requirements phase of development. Further, the methodology extends to the design phase adding a step (step 3) to incorporate a middleware component. We describe each of these steps next.

## 3.1 Step 1: Context-Means Analysis

Context-means analysis (in contrast to means-end analysis) is means of influencing and end. The end can be any of the following.

**Conflicting** goals or plans. For example, the goals *to be tele-reachable* and *to work uninterrupted* in Figure 1 conflict. In other words, a callee cannot possibly accomplish both the goals in a call handling episode. The context of the episode provides a means of capturing which goal the callee might want to accomplish at run time.

**OR decomposition** of goals or plans. While, any child goal in an OR decomposition can satisfy the root goal, given a context, an actor may prefer one child goal (or a subset) over the rest. Context can provide a means of capturing such requirements. The OR decomposition of the plan *to not answer the call* is an example. Note, however, that the decomposed plans in this example are also conflicting.

**Soft goals** where the extent to which a soft goals should be promoted or demoted depends on context. The soft goal *to preserve privacy* is an example. In some cases, context can be used to promote a soft goal and demote another at the same time. The soft goals *to update the caller* and *to preserve privacy* are an example—the more the information a callee gives the better a caller is updated, but the lesser is his privacy preserved.

**Dependencies** where the dependum can be refined to context rather than a goal or a plan. For example, the dependum *to not disturb colleagues* can be refined to context because the actual dependency is that the callee depends on his colleague to provide the context (i.e., whether he can be disturbed or not).

Figure 3 shows the result of context-means analysis of the call handling example from the callee's perspective. Note that a context known to an actor is modeled as a belief. Whereas context as a dependum is modeled as a resource (informational entity). We introduce an additional notation for context-means links.

## 3.2 Step 2: Context Modeling

In the previous step, we found ends where context was a means. Further at each context-means link we used a context abstraction as the place holder for source. In this step, we show how to further elaborate each context abstraction.

First, what does a *context abstraction* represent? The notion on context has been defined by several authors [2]. At a very abstract context is defined as "any information relevant" [3]. While generality is desired for a definition, the current purpose demands specific details. There are two reasons for that. 1) A motivation for system modeling is to document the thought process—the more details the models have the better documented it is. 2) Next, the system specification (derived from the models) is used for software implementation in the last phase of development—the more explicit the model is the easier it is for a developer to implement the software. Thus, the generic notion of context needs to be broken down

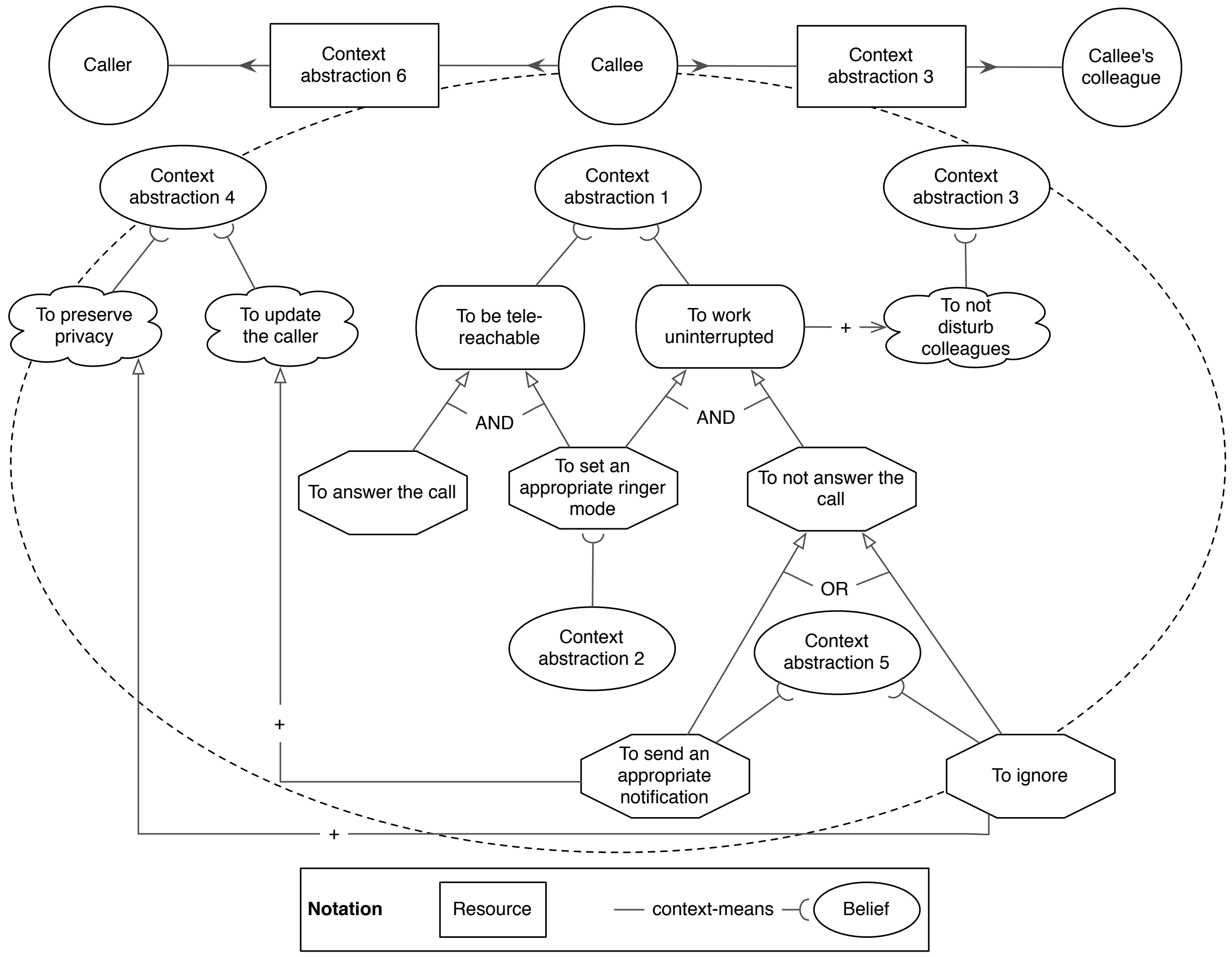

Figure 3: The callee's actor diagram after context-means analysis.

into more specific and meaningful abstractions. We give such a breakdown in Figure 4. This is by no means the only or a complete breakdown. Each node in the figure is a context abstraction, but lower an abstraction in the hierarchy the more specific it is.

Next, what set of abstractions is desired? This of course depends on the end to which the abstraction (or the set) is a means. The crux of this step is to iterate through each abstraction and break it down. Figure 5 shows such a breakdown for context abstractions of the callee. An abstraction should be broken down such that the resulting set of lower-level abstractions cover all situations relevant to the corresponding end. For example, the abstraction context abstraction 1 is a means of handling the conflicting goals of *to be tele-reachable* and *to work uninterrupted*. The breakdown of abstraction into activity and relationship to colleague means that the goal callee chooses to accomplish at run time depends on the callee's activities at that time as well as his relationship with his colleagues. Similarly, the breakdown of context abstraction 4 means that the extent to which the callee wants to update the caller (and preserve privacy at the same time) depends on the callee's relationship to caller and tie

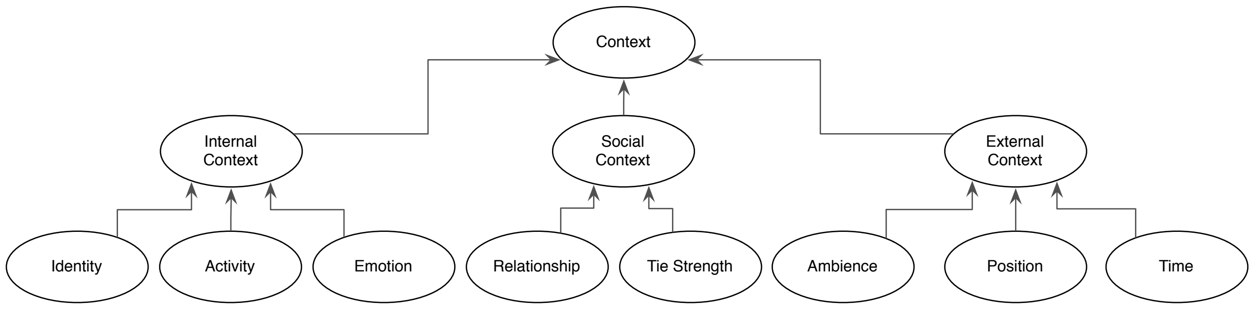

Figure 4: An example context hierarchy (partial). Each node is a context abstraction

strength of caller.

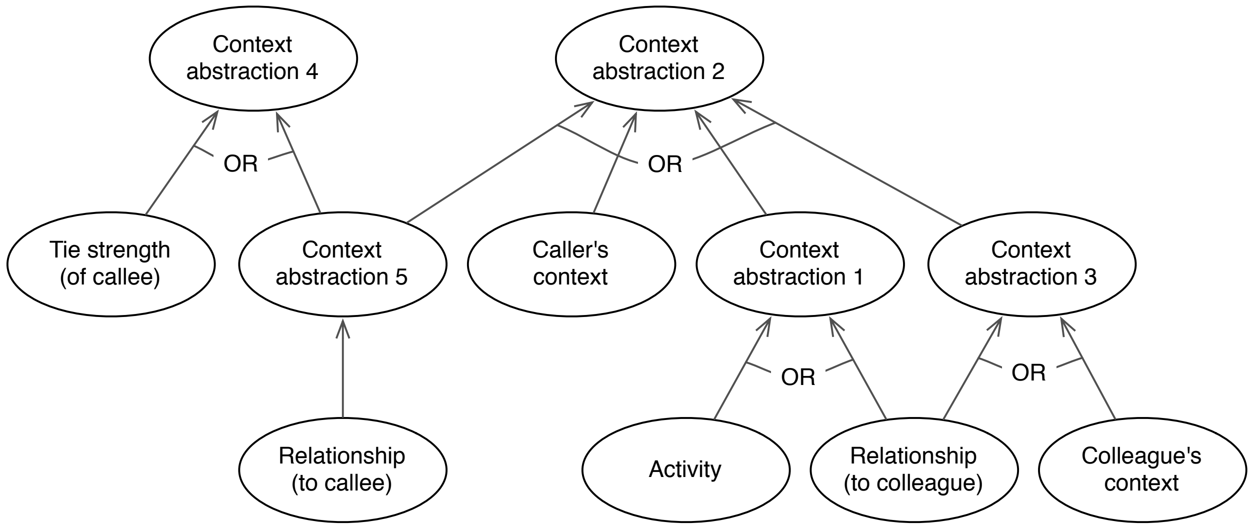

Figure 5: A breakdown of context abstractions of the callee. Each lead node represents a specific abstraction.

Finally, we can model the system-to-be in a similar way—using context-end analysis and context modeling. However, it turns out in this case that the goals of the system to be are essentially the plans of the callee. Thus, the abstractions from callee's perspective can be directly borrowed by the system-to-be. This will lead to the model of system-to-be to look like Figure 6. Note that the soft goals of the system-to-be can now be treated as hard goals since a context-based criteria has now been established.

## 3.3 Step 3: Employing the Platys Middleware

Whereas the previous steps captured the role of context within the system as requirements, the current step is to specify the system from the requirements. The standard Tropos procedure is to map the system-to-be into a set of agents and then to specify a set of

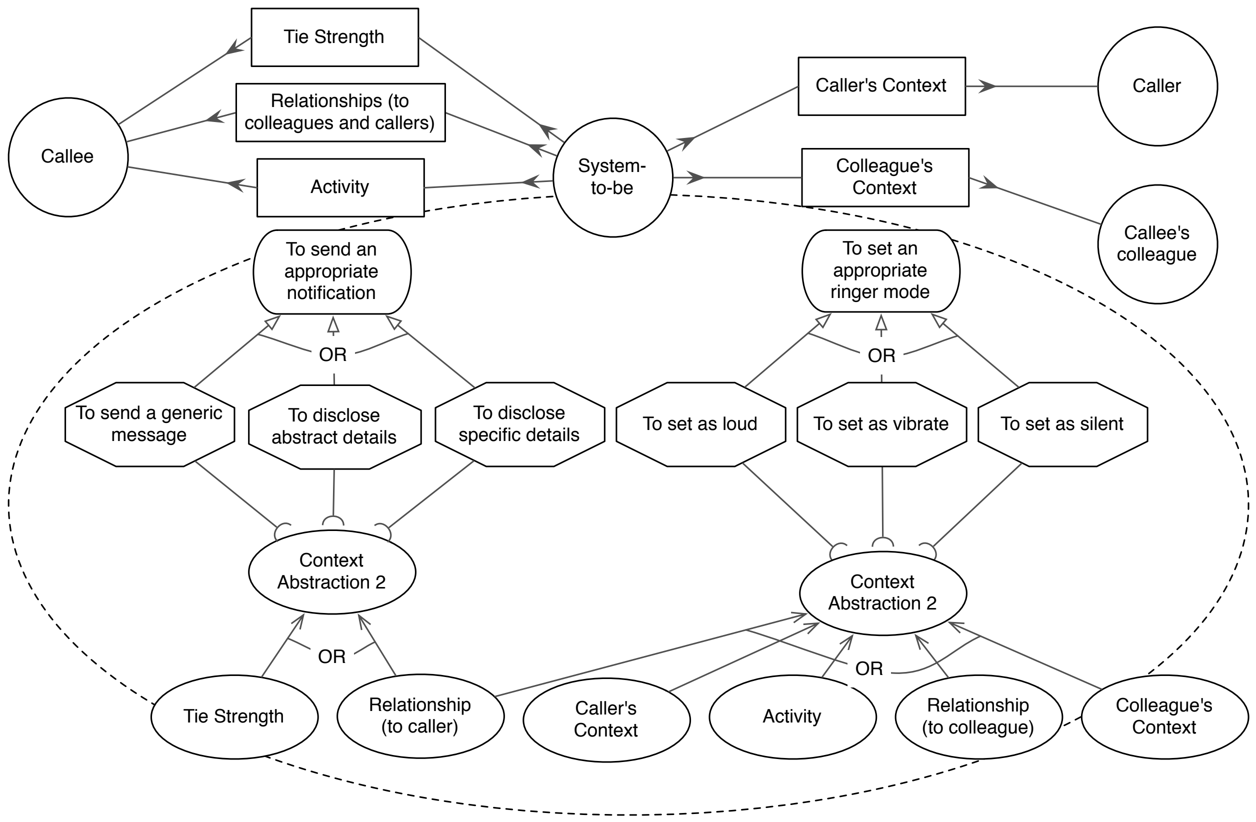

Figure 6: The system-to-be after steps 1 and 2. In contrast to Figure 2 the revised model captures adequate details about the role of context within the system.

capabilities to each agent. While the capabilities required by an agent depends on the goals and plans specific to it, each agent requires the following set of context-specific capabilities.

**To elicit context levels** of each abstraction.

**To learn a context model** corresponding to each abstraction [4].

**To predict context levels** at run time.

**To employ sensors** for learning and prediction.

There we sense an opportunity to incorporate a middleware component into the methodology. The middleware is equipped with the capabilities described above. The advantages of incorporating the middleware are: 1) separation of concerns, 2) less effort for developers, and 3) less effort for end-users. Figure 7 shows the call handling scenario after introducing the middleware.

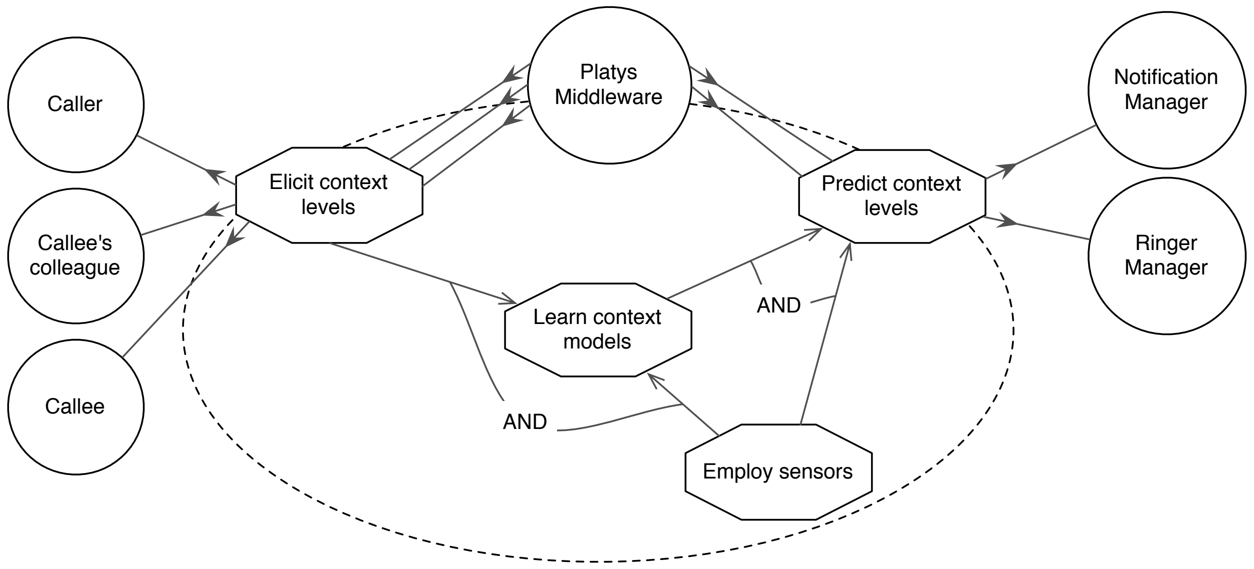

Figure 7: A middleware component is introduced. It is equipped with generic capabilities most context-aware agents require.

## 4 Conclusions

We summarized Tropos and demonstrated the requirements phase of Tropos in a case study. The requirement models can be used to specify the system and consequently implement it. Further, we describe PlatysM, which is an extension of Tropos for capturing contextual requirements. The models in our case study are not necessarily complete. However, it is important to note that there are several ways to model a system and it is largely a creative process.

## 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] Chung-Wei Hang, Pradeep K. Murukannaiah, and Munindar P. Singh. User-centric place identification for location-based services. http://www.csc.ncsu.edu/faculty/mpsingh/papers/tmp/Platys-place.pdf, 2012.