MobiGoal: Flexible Achievement of Personal Goals for Mobile Users

Wenyi Qian, Xin Peng, Huanhuan Wang, John Mylopoulos, Jiahuan Zheng, and Wenyun Zhao

Abstract—Users increasingly depend on mobile applications to get access to software services, social networks, and physical devices. When using mobile applications in their daily lives, users often want to achieve personal goals rather than merely perform individual tasks. To achieve a goal, a user often needs to combine software services, social cooperation from other users, and possibly manual work. Moreover, a goal can often be achieved in different ways, each of which involves an alternative sequence of tasks. Accordingly, mobile applications should be customizable, to accommodate user preferences, and adaptive in changing their configuration automatically if the current configuration is failing. In this paper, we propose an improved runtime goal model that can manage runtime lifecycle of goal instances and adaptively schedule the activation of goals and execution of tasks. Based on the runtime goal model, we propose an agent-based and goal-oriented application framework called MobiGoal, which combines software services, social cooperation, and manual work for achieving user goals and provides an infrastructure for developing customizable and adaptive mobile applications for personal goals. We have developed an implementation of MobiGoal for Android platform and conducted a preliminary empirical study. The results show that MobiGoal applications can effectively support users to adaptively achieve their goals and MobiGoal can significantly save effort of mobile application development for specific goals.

Index Terms—adaptation, service, social cooperation, requirements, goal model, mobile

1 Introduction

Thanks to the wide use of smartphones, users increasingly depend on mobile applications to get access to software services, social network, and physical devices for a variety of purposes such as navigation, electronic commerce, and social interaction. Powered with a growing set of embedded sensors (e.g., accelerometer, GPS, microphone, and camera) [1], today's mobile applications are tightly connected with user behaviors and physical context. Such applications provide location-based and context-aware services [2], [3] that can help users to acquire information or accomplish specific tasks. In the meantime, social computing techniques such as blogs, wikis, and online communities [4], [5] have enhanced mobile applications with social cooperation, thereby enabling collective action and social interaction.

In using such applications, users often want to achieve personal goals rather than merely perform specific tasks. A goal represents a desired state-of-affairs and usually involves a series of tasks. To achieve a goal, a user often needs to combine software services, social cooperation, and manual work. For example, to borrow a book from a library, a user needs to first search for the book and reserve it by invoking a Web service provided by the library, then take the book from the library by himself/herself or delegate the work to a friend. Moreover, a task may be accomplished by either service invocation or social cooperation. For example, a user can reserve a book by either invoking a Web service or asking a friend who is currently near the library to make a reservation on the site. In general, goals can be achieved in different ways, each of which involves an alternative sequence of tasks. For example, a user can acquire a book by borrowing it from a library, or buying it from a book store or a second-hand seller. A user usually has his/her own preferences on alternative solutions for a goal, and at the same time expects that different solutions will be adaptively tried to ensure the achievement of the goal. This kind of flexibility in fulfilling a goal is currently not handled in any systematic way during the development of mobile applications.

Requirements-driven self-adaptation [6], [7], [8], [9], [10], [11] enables a system to adaptively switch among alternative solutions for fulfilling a goal at runtime. These approaches are usually based on runtime goal models. A goal model captures intentions of stakeholders by goals, which are iteratively refined into subgoals/tasks by AND/OR decomposition links. These approaches make adaptation decisions by reasoning over runtime goal models [12], [13], according to the alternative functionalities and their different contributions to relevant quality requirements that are well specified in the models. Some approaches [14], [15], [16] support requirements-driven self-adaptation of socio-technical systems by adopting an agent-based conceptual framework. These approaches, however, are not applicable for developing mobile applications that integrate different means of task accomplishment (service invocation, social cooperation, or manual work) into a unified goal lifecycle management.

In this paper, we propose an agent-based and goal-

- *X. Peng is the corresponding author.*
- *W. Qian, X. Peng, H. Wang, J. Zheng and W. Zhao are with the School of Computer Science and the Shanghai Key Laboratory of Data Science, Fudan University, Shanghai, China.*
  E-mail: {qianwy, pengxin, huanhuanwang13, jiahuanzheng13, wyzhao}@fudan.edu.cn
- *J. Mylopoulos is with the Department of Information Engineering and Computer Science, University of Trento, Italy.*
  E-mail: jm@disi.unitn.it

oriented framework called MobiGoal, which provides a systematic and efficient way for developing mobile applications that can flexibly fulfill personal goals in a customizable and adaptive way with multi-modal task accomplishment. The contributions of this paper are as follows. First, we propose an improved runtime goal model that can manage runtime lifecycle of goal instances and adaptively schedule the activation of goals and execution of tasks. Second, we propose an agent-based and goal-oriented application framework, which combines software services, social cooperation, and manual work for achieving user goals and provides an infrastructure for developing customizable and adaptive mobile applications for personal goals. Third, we have developed an implementation of the framework for Android platform and evaluated its effectiveness with a preliminary empirical study.

With MobiGoal, developers can design a mobile application for a high-level goal by defining a behavioral goal model for it and developing specific software services as required. MobiGoal's runtime goal model includes behavioral information about the fulfillment of goals and a goal state machine for the lifecycle management of goal instances. After obtaining and loading a goal model into a mobile client, a user can customize it into a mobile application for specific goals according to his/her preferences and device situation. The user can customize the priority order of alternative subgoals and the mapping of external events to lifecycle events of goals. Once the root goal is activated, MobiGoal creates an instance of the goal, along with instances for subgoals and tasks, and manages them until the root goal instance is fulfilled. During the process, MobiGoal adaptively schedules the activation of goals and execution of tasks. Moreover, MobiGoal adaptively accomplishes each task by invoking software services, delegating it to other users, or assigning it to the user.

We have developed an implementation of MobiGoal for Android platform using JADE (Java Agent DEvelopment framework) [17]. The implementation includes a mobile client for each user and an agent server. Based on the implementation, we have conducted a preliminary empirical study. In the study, a set of students were requested to customize MobiGoal into a mobile application for book acquisition. The results indicate that MobiGoal applications can effectively support users to adaptively achieve their goals. Our comparative study with ad-hoc application development shows that MobiGoal can significantly save effort of mobile application development for specific goals.

The rest of the paper is structured as follows. Section 2 illustrates the essence of MobiGoal using a motivating example. Section 3 introduces the runtime goal model used in MobiGoal. Section 4 describes the agent-based framework. Section 5 presents our implementation of MobiGoal for Android platform. Section 6 evaluates MobiGoal with an empirical study. Section 7 discusses benefit and possible improvement of MobiGoal. Section 8 reviews related works, while section 9 concludes and outlines future work.

2 Motivating Example

In this section, we illustrate the essence of MobiGoal through a motivating example.

College student Bob often needs to acquire books for his studies. Bob lives on campus where books can be acquired in different ways. They can be borrowed from one of several libraries on campus. They can also be bought from the college book store (CBS) or a second-hand seller (SHS). To buy a book from CBS, a student needs to first find the book in the store, then place an order, and finally get the book. To buy a book from SHS, a student needs to first find the book in the college second-hand market (SHM), then select a seller (also student in the college), and finally trade face to face.

To acquire a book, Bob first tries to borrow it from a library. He uses the website of the college libraries to search for the book and make a reservation. If the online service is not available, he can find a friend who is currently near a library to make a reservation for him. After reservation, he walks to the library where the book is stored to get the reserved book. If he is far from the library, he may want to find a friend who is currently near the library to take the book for him. With the current mobile applications, he needs to use a mobile social networking application like WhatsApp and WeChat to find a friend in his social groups for help. If the book is not available in the libraries, Bob needs to buy it from CBS or SHS. Similarly, he needs to try and switch between different strategies and combine software services, manual work, and social collaboration. During the whole process, Bob needs to manage the states of all the involved tasks and adjust the strategies by himself. Each time when Bob wants to find a friend for help, he needs to choose from a lot of friends who are online based on a series of factors such as degree of intimacy, distance to the target location, and past reputation. After a task is delegated to a friend, Bob needs to wait for his/her message to determine whether the task has been successfully done. An experience like this can be a challenge for Bob, as he has to manage and coordinate every aspect by himself.

Therefore, Bob hopes that he can install an application on his mobile phone to help him acquire books in a flexible way. He hopes that there is no need for him to take care of the details of how to get the book, and the application itself will try different solutions for book acquisition according to his preferences: first tries a library, then a second-hand seller, and finally the book store. The application automatically schedules and monitors the execution process, and adaptively switches among different solutions. For example, after launch, the application first tries to automatically do a library book search and then reserve the required book using Web services provided by his college. If the services are not available or fail, the application tries to find a suitable person from Bob's social network for help or assigns the work to Bob himself. After the book is reserved, the application tries to find a suitable person who is near the target library to get the book from the library or assigns the work to Bob himself. If the application finally fails to borrow the book, it tries to buy the book from a second-hand seller or the book store. Last, but not least, Bob expects that book acquisition solutions embedded in the application can reflect the collective intelligence of his fellow students on campus.

Although Bob's requirements can be satisfied by developing an ad-hoc application, we think that there is a need

for a general framework that can support the systematic development of mobile applications that are customizable and adaptive in the sense described above. The framework can be deployed on mobile devices and customized into applications for different goals.

We propose such a framework called MobiGoal in this paper. In MobiGoal, alternative solutions for achieving a high-level goal are represented by a goal model, which can be shared by users. To achieve the goal of acquiring book, Bob can obtain a goal model for this goal and customize it into a runtime goal model according to his preferences and device situation. Each time the root goal (acquiring book) is activated, for example when Bob clicks a button on the screen, MobiGoal creates a runtime goal model for it and manages the runtime lifecycle of each goal instance. During the execution process, MobiGoal receives external events (e.g., a confirmation of delegated work done from a friend) and schedules the activation and execution of relevant goals and tasks. When a selected solution fails (e.g., a book is not available in the libraries), MobiGoal can switch to another alternative (e.g., buying from CBS). Moreover, MobiGoal can adaptively accomplish a task through assignment to a software service, the user, or a third party. For example, to confirm whether the book store has a required book, MobiGoal first tries to invoke a Web service provided by the store. If the Web service is not available or fails, MobiGoal tries to find a person from Bob's social network who is currently near the store and delegate the task to the person. If Bob is close to the store, MobiGoal may also assign the task to Bob himself by providing task instructions. The results of task execution by social delegation or manual work are captured by user interaction and agent-based communication and are fed back to the lifecycle management of related goal instances. During the process, the application seamlessly integrates user interaction and social cooperation with software services to accomplish the tasks (e.g., book search or book reservation).

3 Background Knowledge

In this section, we introduce the background knowledge of runtime goal model. A runtime goal model consists of a set of goal and task classes, which can be instantiated at runtime and whose runtime instances can be executed. To this end, the concept of behavioral goal model is proposed, which augments basic goal model with runtime behavior annotation that specifies the information about the behaviors through which goals are achieved. On the other hand, a goal state machine is defined to manage the runtime lifecycle of each goal instance by indicating the achievement states of goal instances and transitions between these states.

3.1 Basic Goal Model

In goal-oriented requirements engineering (GORE) [18], goal models are used to capture intentions of stakeholders on the system-to-be and explore alternative ways to satisfy these intentions. In a goal model, stakeholder intentions are represented by goals, which can be refined into subgoals/tasks by AND/OR decomposition links. An AND/OR decomposition means that all/at least one of the subgoals/tasks should be fulfilled to make the parent goal fulfilled. Therefore, goal models can capture alternative ways for fulfilling high-level goals by OR-decompositions. A goal can be iteratively refined until leaf-level tasks can be accomplished by either a software component or a human agent.

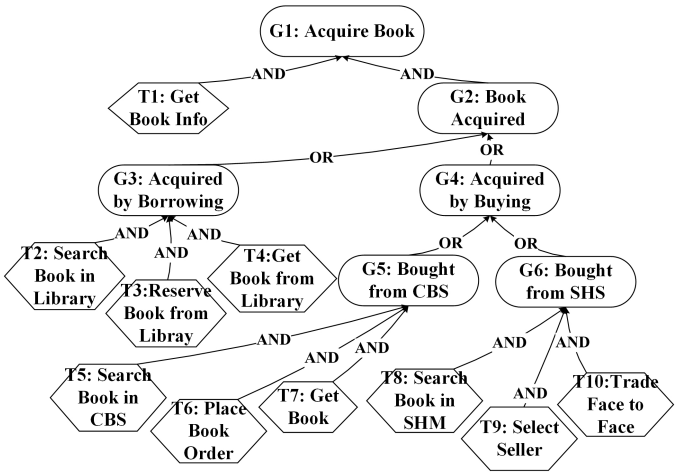


Fig. 1. Goal Model for Acquiring Book

Figure 1 shows an example of goal model for acquiring book, which is introduced in Section 2. It shows the hierarchical refinement of a root goal "Acquire Book" (G1) into subgoals and tasks by AND/OR decomposition links. From the goal model, it can be seen that to fulfill the root goal one needs to first execute the task "Get Book Info" (T1) and then fulfill the subgoal "Book Acquired" (G2). G2 is OR-decomposed into "Acquired by Borrowing" (G3) and "Acquired by Buying" (G4), which means a required book can be acquired either by borrowing or buying it. G4 is further OR-decomposed into "Bought from CBS" (G5) and "Bought from SHS" (G6), which means a required book can be bought either from the college book store (CBS) or a second-hand seller (SHS). These OR decompositions provide alternative strategies for acquiring book, each of which involves an alternative sequence of tasks. For example, the strategy of borrowing a required book from library can be accomplished by executing the task sequence [T1, T2, T3, T4]; the strategy of buying a required book from CBS can be accomplished by executing the task sequence [T1, T5, T6, T7].

3.2 Runtime Behavior Annotation

Runtime behavior annotations specify sequencing and cardinality constraints among goal instances at runtime [13]. Behavior annotation is represented by a formula E (called goal expression) over a set of goal classes. The goal expressions used in MobiGoal and their meanings are listed in Table 1. Each AND- or OR-decomposed goal needs to be annotated with a goal expression specifying the sequencing and cardinality constraints over its subgoals. For example, if an AND-decomposed goal $G1$ requires that one of its subgoals $G2$ be achieved before the start of the other one $G3$, then $G1$ needs to be annotated with an expression $G2; G3$. Goal expression for an OR-decomposed goal indicates the

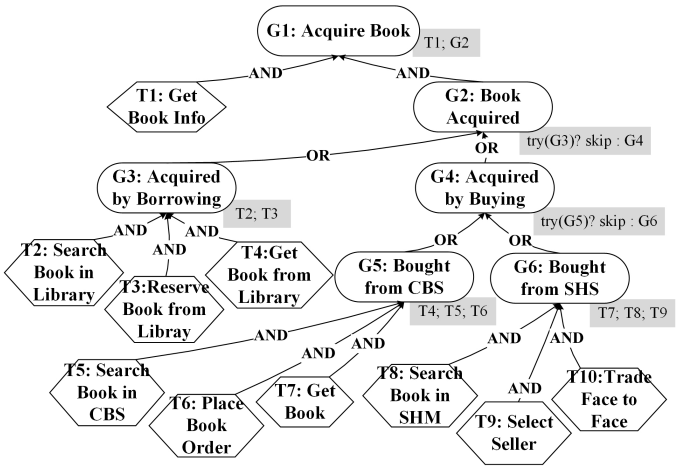


Fig. 2. Behavioral Goal Model for Acquiring Book

TABLE 1
Syntax and Meaning of Goal Expressions

Expression E	Meaning
$skip$	Do nothing
$E_1; E_2$	Sequential occurrence
E^+	One or more sequential occurrences of E (iterated concatenation)
$try(G)?E_1 : E_2$	If goal is achieved, an E_1; otherwise, an E_2
$E_1 \# E_2$	Interleaved occurrence of E_1 and E_2

priority order of its alternative subgoals. For example, for a goal $G1$ with two alternative subgoals $G2$ and $G3$, the expression $try(G2)?skip : G3$ means to try $G2$ first and if it fails try $G3$.

Figure 2 shows the runtime behavior annotations for the goal model for acquiring books. The goal model specifies a goal expression for each AND- or OR-decomposed goal. For example, the goal expression for the AND-decomposed goal G1 indicates that task T1 must be accomplished before the start of goal G2; the goal expression for the OR-decomposed goal G2 indicates that G3 should be first tried and if failed then G4.

3.3 Goal State Machine

At runtime, each goal instance is associated with a goal state machine, which indicates its achievement status. Figure 3 shows the basic goal state machine [15], which includes a series of states of goal instances and transitions between states.

Basic states of a goal instance include **Activated**, **Executing**, **Suspended**, **Achieved**, and **Failed**, indicating that a goal has been committed but not started, is under execution, is suspended, has been achieved, has failed, respectively. Composite state **ShouldDo** indicates that a goal has been activated but not achieved or failed yet.

Transitions between these states are driven by different kinds of events, i.e., *Activate*, *Start*, *End*, *Suspend*, and *Resume*. These events can be mapped to external events, for example, the click of a button on the screen or the arrival of

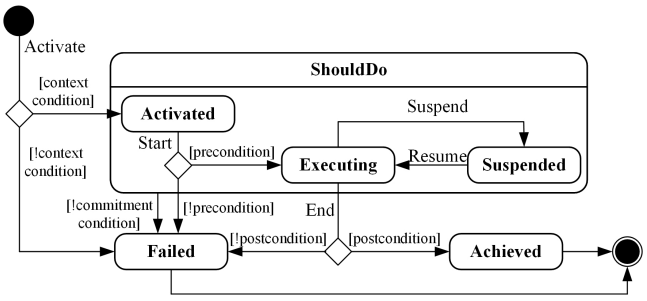


Fig. 3. Goal State Machine

specific point of time every day, or internal events that reflect interactions between parent goals and subgoals/tasks. If a lifecycle event is mapped to null, then it means that the state transition can occur immediately after the end of the source state. For example, if the *Start* event of a goal is mapped to null, it means that an instance of the goal is started immediately after it is activated.

Some of the transitions are associated with conditions, including *context condition*, *precondition*, *postcondition*, and *commitment condition*. Context condition expresses whether the goal is applicable [9], and is checked before a goal is activated. Precondition specifies the condition that should be satisfied when a goal starts to execute. Sometimes a precondition is not satisfied currently, but may be satisfied later. Postcondition specifies the condition that should be satisfied before a goal is achieved. Commitment condition determines a time limit within which a goal is committed to be achieved, and is continuously checked during **ShouldDo** state. For example, the context condition "The user is authorized as legal" can be applied to the root goal "Acquire Book" (G1) to specify the user should be authorized; The precondition "The inventory is sufficient" and the postcondition "The order has been created in database" can be applied to the task "Place Book Order" (T6).

4 Agent-Based Framework

This section first presents an overview of the agent-based framework, and then details two key parts of it, i.e., runtime goal model management and task execution.

4.1 Overview

The proposed agent-based framework includes an agent server and a client installed on the mobile device of each user. Figure 4 presents an overview of the framework, showing the main modules on mobile clients and the agent server and the interactions between these modules.

A mobile client manages runtime goal models of a user and helps to achieve his/her goals adaptively. The agent server maintains a repository of shared goal models and a list of online clients. To interact with other mobile clients, a mobile client needs to register itself with the agent server and discover other users through the server. It periodically sends a heartbeat message to the agent server to indicate the availability and current position of the user, then it can be discovered by other users through the server. To avoid being disturbed or protect privacy, a user can set his/her

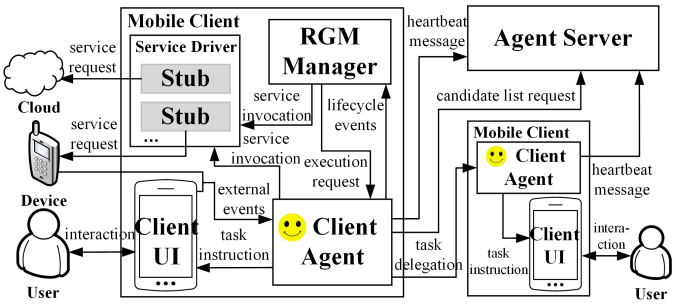


Fig. 4. Overview of Agent-based Framework

TABLE 2
Parameter Passing Rules for Acquiring Book

Parameter	Type	Produced by	Consumed by
bookTitle	String	T1	T2,T4,T7
ISBN	String	T2,T4	T3,T5,T6
libraryLocation	Location	T2	T3
bookPrice	Double	T4	T5,T6
sellersInfo	List	T7	T8
sellerInfo	String	T8	T9
sellerLocation	Location	T8	T9

mobile client to be offline or only receive cooperation invitations meeting certain conditions. For example, the mobile client can be set to only receive invitations from specific user group or for specific tasks, or for tasks whose target locations are within a certain distance.

To achieve a high-level goal G, a user needs to obtain a corresponding goal model (with G as its root goal) from the server and download it into his/her mobile client. The goal model specifies fulfillment strategies for G with variation points represented by OR decompositions. In addition, the user needs to customize the priority order of alternative subgoals and the mapping of external events to lifecycle events of goals.

The main modules on mobile clients include RGM (Runtime Goal Model) manager, client UI (User Interface), service driver, and client agent. The RGM manager maintains a runtime goal model for each activated high-level goal and manages the runtime lifecycle of goal instances. The service driver provides service stubs to software services (e.g., web services) that can be accessed on the device. The client UI provides user interfaces for goal management and manual task execution.

The client agent communicates with the agent server and the client agents of other users. It periodically sends a heartbeat message to the agent server. It also coordinates the interactions between the client agent, the service driver, and the client UI. The client agent captures registered external events (e.g., click of a button on the client UI) and passes them to the RGM manager. It coordinates the execution of tasks in different manners, including automatic execution by invoking software services, manual execution by assigning work to the user, and social execution by delegating tasks to others.

Note that adaptation is supported at two levels during the process of goal achievement. At the goal level, the RGM manager adaptively achieves a goal by switching among alternative solutions of it. At the task level, the client agent adaptively accomplishes a task by trying different ways (automatic, manual, or social execution). These two key parts of the approach are detailed in Section 4.2 and Section 4.3 respectively.

4.2 Runtime Goal Model Management

In this subsection, we first introduce the runtime goal model used in MobiGoal, and then user customization of goal models and the goal lifecycle management at runtime.

4.2.1 *Extended Runtime Goal Model*

The runtime goal model used in MobiGoal extends the behavioral goal model and the goal state machine introduced in Section 3.

The runtime goal model extends behavioral goal model with parameter passing. Similar to data flow analysis, there is also data shared among different tasks. During the process of goal achievement some data may be shared among different tasks. Data produced by some tasks will be consumed by other tasks. This kind of data flow relations are described as parameter passing rules among different tasks. Table 2 lists the parameter passing rules for the goal model shown in Figure 2. Each rule specifies the name, type, producer and consumer of a parameter. For example, the rule for $libraryLocation$ specifies that the library location produced by "Search Book in Library" (T2) will be consumed by "Borrow Book from Library" (T3). Moreover, data parameters produced by tasks can also be used as context variables in goal state transition condition expressions (see Section 3.3).

The goal state machine used in MobiGoal is shown in Figure 5. It extends the goal state machine introduced in Section 3 and the one proposed in our previous work [15] from two aspects. One is new temporary states for facilitating condition checking and self-repairing. The other is top-down activation and execution scheduling of goal and task instances.

The three newly introduced temporary states are:

- **Waiting** means an activated goal is waiting for its precondition to be satisfied;
- **ProgressChecking** means checking whether a goal has been achieved when receiving *SubAchieved* event from its subgoals or tasks;
- **Repairing** means a goal is under repairing for some problems such as subgoal failure.

The conditions associated with different transitions are also the same as those shown in Figure 3, except that if the precondition of a goal is violated, further action depends on whether it is waitable or not. A waitable condition means that a condition can be satisfied after a while even it is violated currently. Therefore, if the precondition is waitable, the goal will wait for it to be satisfied until waiting timeout is reached, otherwise the goal is failed. All the conditions in the goal state machine can be specified as a condition expression involving context variables. For example, the

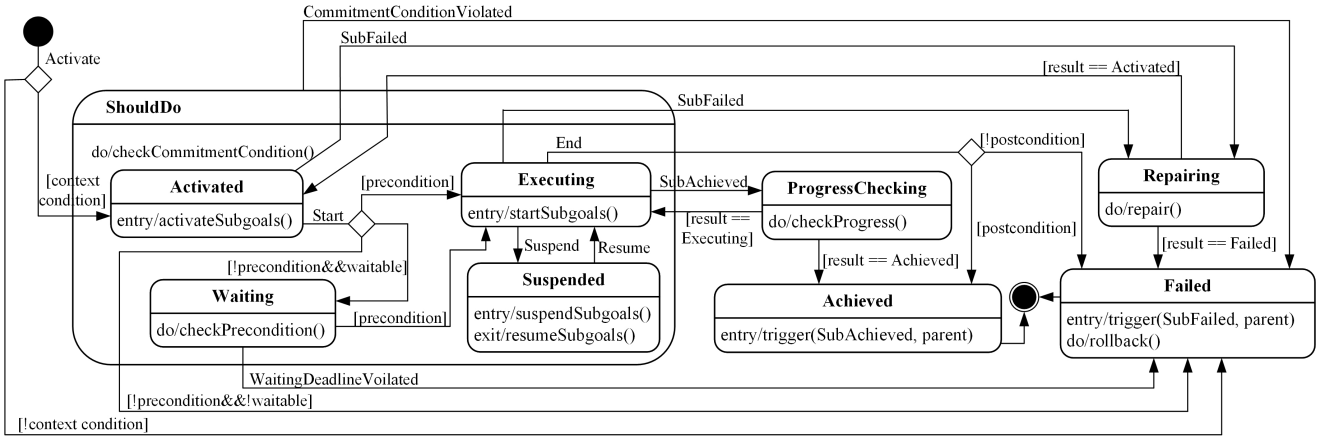


Fig. 5. Goal State Machine

precondition for borrowing book from library can be expressed as "$lib.isOpen$=true". Note that it is not necessary to define all these conditions for all the goals in a goal model. For example, commitment condition usually is only required for the root goal. An undefined *context condition*, *precondition*, *postcondition*, or *commitment condition* is always treated as true at runtime.

More kinds of events are introduced to express the interaction between different goal state machines, i.e., *SubAchieved*, and *SubFailed*. These two events, together with the existed events, i.e., *Activate*, *Start*, *End*, *Suspend*, and *Resume*, are all called lifecycle events. There are two kinds of lifecycle events, i.e., external events and internal events. External events are events obtained from external sources, for example, activating a goal with the click of a button or the arrival of a specific point in time. Mapping rules for these events can be specified in runtime goal model definition and changed during user customization (see Section 4.2.2). Internal events are events sent and received between the goal state machines of a goal and its subgoals/tasks. Internal events include scheduling events sent from a goal to its subgoals/tasks and execution feedback events sent from a goal/task to its parent goal. For example, a goal activates one of its subgoals by sending an *Activate* event to it; a goal sends a *SubAchieved* event to its parent goal when the goal is achieved. Internal events reflect interactions between different goal state machines and are embedded in corresponding state actions. For example, the entry actions of **Achieved** and **Failed** respectively trigger a *SubAchieved* event and a *SubFailed* event to the parent goal, while other scheduling events are embedded in more complex state actions such as $activateSubgoals()$ (see Algorithm 1) and $startSubgoals()$ (see Algorithm 2).

$activateSubgoals()$ is executed when a goal instance enters **Activated** state. It activates all the subgoals/tasks if the goal instance is AND-decomposed or a next subgoal/-task selected by priority order if the goal instance is OR-decomposed. If the goal instance is OR-decomposed and repaired from an execution failure, a *Start* event will be triggered for itself to immediately start its execution.

$startSubgoals()$ is executed when a goal instance enters **Executing** state. For an AND-decomposed goal, it starts the execution of all its subgoals/tasks with no external *Start* events if its subgoals/tasks are in interleaved occurrence, or the first subgoal/task in sequential order if its subgoals/tasks are in sequential occurrence. For an OR-decomposed goal, it starts the execution of the currently activated subgoal.

Algorithm 1 Activate Subgoals

```
1:  procedure ACTIVATESUBGOALS()
2:      if this.isANDDecomposed then
3:          for each sub ∈ this.subgoals do
4:              trigger(Activate, sub)
5:          end for
6:      else if this.isORDecomposed then
7:          sub = this.getNextSub()
8:          trigger(Activate, sub)
9:          if this.repairedFromExeFailure then
10:             trigger(Start, this)
11:         end if
12:     end if
13: end procedure
```

Algorithm 2 Start Subgoals

```
1:  procedure STARTSUBGOALS()
2:      if this.isANDDecomposed then
3:          if this.isInterleaved then
4:              for each sub ∈ this.subgoals do
5:                  if !sub.hasStartEvent then
6:                      trigger(Start, sub)
7:                  end if
8:              end for
9:          else if this.isSequential then
10:             sub = this.getFirstSub()
11:             trigger(Start, sub)
12:         end if
13:     else if this.isORDecomposed then
14:         sub = this.getCurrentSub()
15:         trigger(Start, sub)
16:     end if
17: end procedure
```

Repairing action (see Algorithm 3) is executed when a *SubFailed* event is received from a subgoal/task of the current goal instance. For an AND-decomposed goal, it simply returns $Failed$. For an OR-decompose goal, if a next subgoal/task cannot be found (i.e., all its subgoals/tasks have been failed), it returns $Failed$; otherwise, it returns $Activated$ and sets a flag to indicate whether the goal instance is repaired from an execution failure or not. In either case, the current goal in turn triggers a *SubFailed* event for its parent goal if the repairing action returns $Failed$.

Algorithm 3 Repair Goal

```
1: function REPAIR()
2:     if this.isANDDecomposed then
3:         return Failed
4:     else if this.isORDecomposed then
5:         if this.getNextSub() == null then
6:             return Failed
7:         else
8:             if this.lastState == Executing then
9:                 this.repairedFromExeFailure = true
10:            else
11:                this.repairedFromExeFailure = false
12:            end if
13:            return Activated
14:        end if
15:    end if
16:    return Failed
17: end function
```

4.2.2 User Customization

User customization of goal models includes the priority order of alternative subgoals of OR-decomposed goals and the mapping of external events to lifecycle events of goals.

Priority order determines the activation order of the subgoals of an OR-decomposed goal. For example, in Figure 2, "Book Acquired" (G2) is annotated with "try(G3)?skip:G4", which indicates the priority order of first trying G3 and then G4 for G2. For each OR-decomposed goal, the user needs to specify a sequential order of all its subgoals as their priority order.

The user can customize the lifecycle events of specific goals by mapping registered external events to them. External event customization reflects when the user wants to activate a goal, or start, suspend, or resume its execution. According to the behaviors of runtime goal model (see Section 3), the user must define an external event mapping for the *Activate* event of the root goal and can optionally define external event mappings for the other lifecycle events of the goals in a goal model.

For a lifecycle event of a goal, the user can define its external event mapping based on predefined event types. MobiGoal supports three kinds of external events: *Timer Event* means events related to timer, e.g., the arrival of a specific point in time; *UI Event* means events produced by the client UI, e.g., the click event of a button; *System Event* means events distributed by the operating system (e.g., Android) running on the user's mobile device, e.g., an incoming call. For example, the *Activate* event of the root goal "Acquire Book" (G1) in Figure 2 can be customized to the click event of a button on the client UI; the *Activate* event of a root goal for everyday exercise can be customized to a fixed time every day. Each event type has a set of attributes and the user can precisely specify an external event based on these attributes. For example, an incoming call event can be specified by the caller number; a button click event can be specified by the button name.

4.2.3 Goal Lifecycle Management

A runtime goal model includes an instance (with an associated goal state machine) for each goal/task defined in the corresponding goal model. The RGM manager manages the lifecycle of these goal instances by event mapping, state transition, and parameter passing.

When receiving external events from the client agent, the RGM manager maps them to the lifecycle events of specific goals (e.g., the start event of a goal) according to user defined mapping rules. These events then drive the state transitions of corresponding goal instances. Besides external events, there are also internal events that reflect interactions of different goal state machines. These events are sent and received between instances of goals and their subgoals or tasks (see Section 4.2.1).

When receiving an external or internal event, a goal instance performs state transition according to the definition of goal state machine (see Figure 5). To check a state transition condition (e.g., context condition or precondition), the RGM manager evaluates the value of its condition expression. Context variables involved in a condition expression are resolved by invoking corresponding software services via the service driver (see Section 4.3.1) or reading values of data parameters produced by task execution. For example, the value of a context variable *Current Temperature* can be obtained by invoking a sensor service provided by temperature sensor or a web service for local weather report.

The RGM manager performs parameter passing among different tasks according to the parameter passing rules defined for the goal model. It maintains a parameter passing table as shown in Table 2. When a task is finished, the RGM manager collects the values of its output parameters and stores them in the table. These values will be retrieved by the client agent and used as input parameters when executing a task. For example, according to the rules defined in Table 2, the parameter $bookTitle$ obtained from T1 will be used as an input parameter for T2, T4 and T7.

4.3 Task Execution

When a task in the goal model is scheduled to execute, the RGM manager sends a task execution request to the client agent. Then the client agent tries to accomplish the task in different ways. The client agent first tries to automatically accomplish the task by invoking software services. If automatic execution fails, the client agent will try to assign the task to the user himself/herself or delegate the task to other users.

Sometimes, a task may be executed only by delegation or only by software service invocation. For example, if there needs an external parameter that should be input by the

user, such kind of tasks can only be delegated. Or the task is too complicated for human beings, then it can only be accomplished by service invocation.

After the task is successfully accomplished, the client agent will return the result (including output parameters) to the RGM manager, which in turn will send an *End* event to corresponding goal instance. If all the task execution attempts have failed, a task execution failure will be sent to the RGM manager, which in turn will trigger goal repairing mechanisms.

4.3.1 Service Invocation

Invocation of software services is supported by the service driver in mobile client. The service driver maintains a list of software services that can be accessed on the current mobile device and provides a service stub for each service. MobiGoal supports three kinds of software service, including web service, device service and context service. Web service is a remote service that can be accessed via the Internet. Device service is a local service provided by the user's mobile device through the applications installed on it. For example, an API (application programming interface) provided by an application or a device sensor can be wrapped as a device service. Context service is a service that is provided by devices and sensors in the context and can be accessed via near field communication or bluetooth. For example, a bluetooth controller for an air conditioner can be wrapped as a context service. Note that context service is usually location based, which means its availability may be distance-dependent. For each registered software service, a service stub is maintained by the service driver, which provides a standard local interface.

In order to invoke a software service, the client agent will first initiate an asynchronous execution thread. The execution thread then sends a service request to the service driver and obtains an ordered list of matched service stubs from it. After that the client agent invokes through the standard interface each of the obtained service stubs in turn until the task is successfully accomplished. If all the service invocations fail, the execution thread will return a failure to the client agent, and the client agent will turn into trying manual work or social delegation to accomplish the task.

4.3.2 Manual Work & Social Delegation

When trying manual work or social delegation to accomplish a task, the client agent requests a candidate user list from the agent server. Agent server selects candidate users from those who are online and near the target location of the task. Selected candidates are ranked based on a combined utility of intimacy (degree of intimacy with the current user), reputation (reputation level in past experience), and proximity (proximity to the target location) with the following equation:

$$Utility = w_i * V_i + w_r * V_r + w_p * V_p$$

where V_i, V_r, V_p are the intimacy, reputation, and proximity respectively; w_i, w_r, w_p are the weights of them which satisfy $w_i + w_r + w_p = 1$. We assume that the intimacy V_i can be obtained based on the user groups and tags of the candidate user; the reputation level V_r can be computed based on the records of the candidate users past experience of social cooperation, for example, by success rate. The proximity V_p can be computed as the inverse function of distance to the target location. All these three parameters are variables between 0 and 1 and a bigger value means greater intimacy, reputation and proximity respectively. Note that the current user himself/herself is also treated as a candidate with intimacy and reputation equaling 1.

Based on the returned ranking of candidate users, the client agent chooses the highest-ranked one. If the chosen one is the user himself/herself, the client agent generates a task instruction with task description and input parameters and sends it to the client UI. The client UI then shows the task instruction to the user and provides an interaction form to collect output parameters. When the user finishes the task, the client UI sends a task finished message with returned output parameters to the client agent. If the chosen one is another user, the client agent sends a task delegation message to the user's client agent. Then similar to task assignment to the current user, the client UI of the delegated user shows the task instruction and provides an interaction form. If the delegated user agrees to accept the delegation, he/she finishes the task and his/her client agent sends a task finished message with returned output parameters to the client agent of the current user. If the delegated user fails or refuses to accept the delegation, the client agent chooses the next candidate to accomplish the task until the task is successfully accomplished.

5 Implementation

We have developed an implementation of MobiGoal for Android platform. Each goal model is described as an XML file and shared on the server. A part of the XML schema is shown below.

```
<xsd:element name="GoalModel">
   <xsd:complexType>
   <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
      <xsd:element name="goal" maxOccurs="unbounded">
         <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="id" type="xsd:string"/>
            <xsd:element name="name" type="xsd:string"/>
            <xsd:element name="parent" type="xsd:string"/>
            <xsd:element name="decomp" type="xsd:string"/>
            <xsd:element name="exp" type="xsd:string"/>
         </xsd:sequence>
         </xsd:complexType>
      </xsd:element>
      ...
```

A goal model consists of a series of goals and tasks. For each goal or task, it should specify the id, name, parent goal ($NULL$ for the root goal), decomposition type (AND or OR, and $NULL$ for a task), and behavioral expression ($NULL$ for a task) of the goal. For example, a fragment of the XML description of the goal model for acquiring book is presented below.

```
<GoalModel>
   <name>Acquiring Book</name>
   <goal>
      <id>G1</id>
      <name>Acquire Book</name>
      <parent>NULL</parent>
      <decomp>AND</decomp>
      <exp>T1;G2</exp>
   </goal>
```

```
<goal>
   <id>T1</id>
   <name>Get Book Info</name>
   <parent>Acquire Book</parent>
   <decomp>NULL</decomp>
   <exp>NULL</exp>
</goal>
......
```

When a goal model is downloaded into a mobile client, it is parsed for user customization. Once the root goal of a customized goal model is activated, the mobile client creates an instance of the goal model and manages its lifecycle.

At runtime, each goal instance is represented by a Java object with an embedded goal state machine running on a separate thread. Each goal instance independently manages its own state, receives external events distributed by the RGM manager, and interacts with other goal instances. The RGM manager implements a message communication mechanism to facilitate external event distribution and goal instance interaction. Each message has a **Sender** and a **Receiver** to specify the goal instances that send and receive the message respectively, and a **Type** to specify the type of lifecycle event (e.g., *Activate*, *SubAchieved*).

The client agent and agent server have been implemented using JADE [17], which is an open source platform for peer-to-peer agent based applications. The client agent is implemented as a JADE agent and runs as an Android Service on mobile device. Communication from Java object to agent, for example an execution request sent from the RGM manager to the client agent, is implemented by using *O2AInterface* provided by JADE. Communication between different agents is implemented by using ACL (Agent Communication Language). To capture system events distributed by Android system, the client agent registers a *BroadcastReceiver* for each Android event type that is defined as an external event. For example, to capture an incoming call event, the client agent registers a *BroadcastReceiver* with $ACTION\_NEW\_INCOMING\_CALL$ and $PHONE\_STATE$ action filters.

Each service stub in the service driver is implemented as an Android *IntentService*, which runs an operation on a single background thread. Thus, a software service can be invoked in an asynchronous way when being used to accomplish a task. Service stubs can be obtained on demand from the server and dynamically loaded into the mobile client.

Figure 6 shows the user interfaces of the mobile client, including the following views: **MyGoal** (Figure 6(a)) view lists all the goal models that have been customized and loaded; **Message** (Figure 6(b)) view shows work assignment messages and task delegation messages; **Execution** (Figure 6(c)) view lists all the finished or ongoing execution instances of goal models; **Procedure** (Figure 6(d)) view shows the procedure of an execution instance of a goal model; **Progress** (Figure 6(e)) view shows the current state of each goal/task of a goal model in an execution instance.

In **MyGoal** view, the user can check the structures and behavior information of existing goals models. He/she can also view and modify the customization of each goal model. To obtain a new goal model, the user can click the plus button to check and download goal models shared on the server.

In **Message** view, the user can click a work assignment or task delegation message to check the task instruction. For a task delegation message, the user can decline the delegation by clicking a "Decline" button. The task instruction UI provides instruction for the user to accomplish a task and an interaction form for the user to input required results. It is automatically generated based on an XML-based task UI description given in the goal model definition. Currently, our implementation supports different kinds of fields, including text input, drop-down box, picture box, and confirmation button. For example, for a task "Borrow Book from Library" (T3) assigned by the client agent or delegated by other users, task instruction UI shows the book title and library name. The interaction form includes a "Done" button and a "Fail" button for the user to confirm the accomplishment or failure of the task.

In **Execution** view, each execution instance of goal models is displayed as an item with its title, activation time, and status (achieved, failed, or executing). To manually initiate an execution instance, the user can click the plus button and select an existing goal model to activate its root goal. Clicking an execution instance opens a **Procedure** view showing the execution procedure of it. The procedure consists of a sequence of external event logs, each of which records the occurrence of a lifecycle event (e.g., the activation or start of a goal) or the task execution information (e.g., invoking software service or delegating the task). Clicking the gear icon in **Procedure** view opens a **Progress** view, which shows the state of each goal/task in the goal model: filled circle means **Activated**; hyphen means **Executing**; tick means **Achieved**; cross means **Failed**; exclamatory mark means **Suspended**.

6 Empirical Study

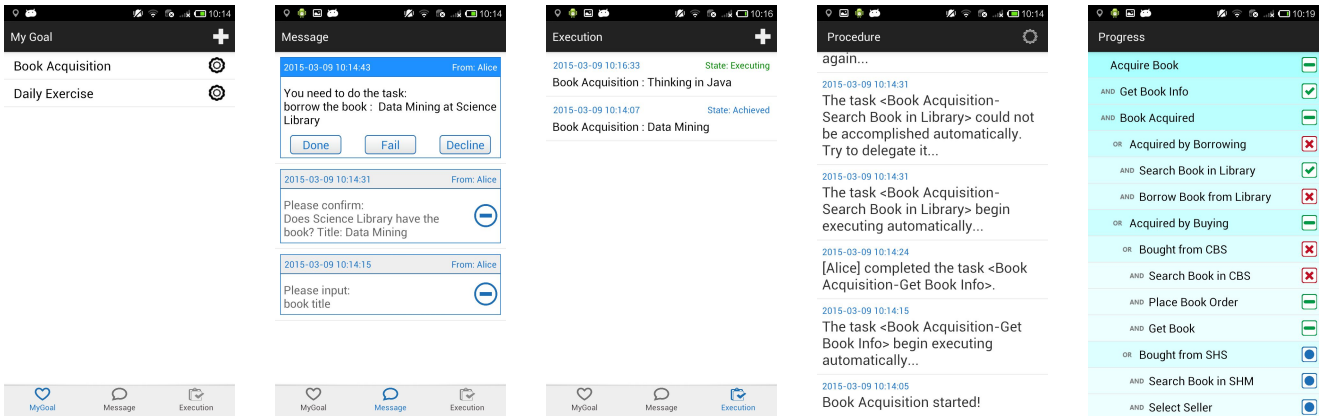
To evaluate the effectiveness of MobiGoal, we conducted a preliminary empirical study based on our implementation for Android platform to answer the following three research questions:

- **RQ1**: Can customized MobiGoal applications effectively support users to achieve their goals adaptively in real life?
- **RQ2**: How is the user experience of MobiGoal applications? What are the main advantages of MobiGoal and how can it be further improved from the point of view of the users?
- **RQ3**: How much effort can MobiGoal save for developing customizable and adaptive mobile applications compared with ad-hoc application development?

6.1 Settings

For **RQ1**, we conducted a user study with the goal model for acquiring book (see Figure 2). The *Activate* and *Start* events of the root goal were mapped to click events of corresponding buttons on the client UI. The default priority orders were: G3 over G4, G5 over G6.

We recruited 14 master students from the school of computer science of Fudan University. Seven of them were requested to customize a book acquisition application based on the given goal model and use the application to acquire



(a) MyGoal View (b) Message View (c) Execution View (d) Procedure View (e) Progress View

Fig. 6. User Interfaces of Mobile Client

books in a simulated campus environment. The other seven participants were requested to play different roles in the simulated environment: two librarians in two libraries, one clerk in a college book store (CBS), two second-hand sellers (SHS), and two passerby pedestrians. Each of these simulated roles except for the two pedestrians was located at a fixed position on campus.

The seven MobiGoal users and the two pedestrians installed the mobile client of MobiGoal on their own mobile phones. Each time a user used the MobiGoal application to acquire a book, the other eight participants walked around campus as candidates for task delegation. We deployed some software services such as a library book search service for automatic task execution and simulated some uncertainties for task execution. For example, some books were not in the two libraries but could be bought from a book store or a second-hand seller; the library book search service was unavailable in some cases. We collected and analyzed the book acquisition processes and results of the seven users.

For **RQ2**, we conducted a post-experiment survey to get user feedback on the MobiGoal application for acquiring books. Moreover, we conducted a group interview. Based on the survey and interview, we summarize the user experience of the application and the users' improvement suggestions for MobiGoal.

For **RQ3**, we conducted a comparative study between MobiGoal-based application development with ad-hoc development. A master student, who had one year of experience in Android application development, was asked to develop the same book acquisition application in two different ways. She first developed the application using MobiGoal by defining a behavioral goal model and default settings such as priority orders. Then she developed an ad-hoc Android application that implements the same book acquisition functionalities and supports user customization and runtime adaptation. Note that all the software services (e.g., book search services) and service stubs are all developed in advance and provided for application development. Moreover, the ad-hoc development directly used APIs provided by the agent server (e.g., candidate discovery and ranking, delegation request) to implement task delegation. We compared the development effort in terms of development time and lines of code.

TABLE 3
Results of User Study (C.T. for Customization Time, E.T. for Execution Time, G.A. for Times of Goal Adaptations, T.A. for Times of Task Adaptations)

User	Achieved	C.T.(s)	E.T.(s)	G.A.(#)	T.A.(#)
User 1	Yes	50	135	2	0
User 2	Yes	1	189	2	4
User 3	Yes	88	145	0	1
User 4	Yes	0	418	2	6
User 5	Yes	178	127	0	0
User 6	No	26	286	2	6
User 7	Yes	44	128	1	0

6.2 RQ1: Effectiveness

Among the seven users, four of them followed the default priority settings in user customization. The other three changed the priority settings to G4 over G3 or G6 over G5. Table 3 shows the results of user study. The six columns respectively list the user, final result (root goal achieved or not), customization time in seconds (C.T.(s)), execution time in seconds (E.T.(s)), times of goal adaptations (G.A.(#)), times of task adaptations (T.A.(#)).

From the table we can see that all the users except User 6 finally achieved their goals (i.e., acquired books successfully). The users spent different time on user customization. User 2 and User 4 followed the default settings and spent nearly no time on customization. The other users spent some time on customizing priority orders of alternative subgoals/tasks. The users spent two to seven minutes (execution time) to achieve their goals. All the users except User 5 experienced at least one goal or task adaptation. After analyzing the processes, we found that the length of execution time usually depended on the number of goal/task adaptations and the availability of candidates for task delegation near the target locations.

Table 4 shows the execution process of each user. The first column shows the user and the second column shows his/her execution process as a trace of goal/task execution.

For simplicity, we do not show the intermediate process (e.g., *Start*, *Repair*) of a goal/task execution. Instead, we only show the activation of a goal (italic) and the success (bold) and failure (hollow) of a goal/task. In addition, the subscript of a task shows how it was executed: "I" means the task was automatically executed by invoking services, "D" means the task was executed by delegating it to others, and "M" means the task was manually executed by assigning the work to the user.

From the table we can see that User 5 successfully achieved his goal of acquiring required book without any adaptation. All subgoals were activated and achieved according to the priority orders set by him and all relevant tasks were accomplished smoothly. User 6 requested a book that was not available in the experiment, so he finally failed to get the book after trying all the three ways of acquiring books.

All the other users successfully achieved their goals by goal/task adaptations. For example, for User 2, as the priority orders set by him, MobiGoal first activated G3 (Acquired by Borrowing) and executed T2 (Search Book in Library) by invoking the Web service provided by the library. But unfortunately, the book search service was unavailable at that time, so MobiGoal tried to accomplish T2 by delegation. MobiGoal delegated the task to four users near the library successively, but none of them could find the book and G3 failed. Then MobiGoal switched to G4 (Acquired by Buying) and activated G5 (Bought from CBS). Next MobiGoal successfully accomplished T5 (Search Book in CBS) by invoking the CBS book search service, but failed to accomplish T6. Finally, MobiGoal switched to G6 (Bought from SHS) and several second-hand sellers were found by executing T8 (Search Book in SHM). User 2 was instructed to select one seller from the sellers (T9) and get the book through face-to-face trading (T10).

In summary, our studies offer positive evidence that the customized MobiGoal application for acquiring book can effectively support users to achieve their goals adaptively. Only User 6 failed to acquire his book in the experiment, but his failure was due to the fact that the book he required was not available at the time of the experiment.

6.3 RQ2: User Feedback

In the post-experiment survey, the seven users were asked to rate the functional and non-functional aspects of the application with a score between 1 to 5 (1 the lowest, 5 the highest) and gave their feedback on the application. The functional aspect refers to the functional design of MobiGoal applications for the achievement of user goals. The non-functional aspect refers to the non-functional quality attributes of MobiGoal applications, such as reliability and usability.

Figure 7 shows the scores given by each user. For the functional aspect, three users gave a score of 5 and the other four gave a score of 4, making an average score of 4.4. In general, the users were satisfied with the scheduling and adaptation of goal achievement strategies and task execution. Even User 6, who failed to get the required book in the experiment, gave a score of 5. He explained that, although he failed to get the book, he realized the adaptation process

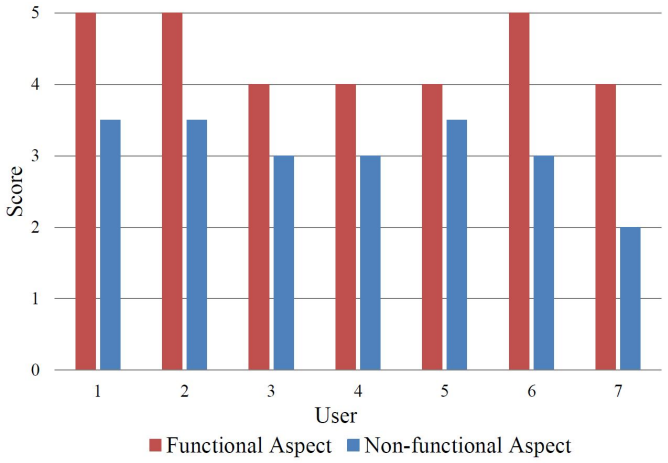


Fig. 7. Scores of Functionality and Non-functionality of MobiGoal

that MobiGoal had tried for his goal after checking the **Procedure** view of MobiGoal. He believed that MobiGoal provides a convenient and effective way for users to achieve their personal goals.

For the non-functional aspect, the average score is 3.1 with the highest score of 3.5 from User 1, 2, 5 and the lowest score of 2 from User 7. The concerns of the users are mainly on the reliability and usability of MobiGoal applications. The reliability concern is highly related to the fact that MobiGoal is sensitive to the network condition in task delegation. In some cases, MobiGoal may delegate a task to a user who was temporarily unreachable due to disconnection from the campus network. This makes the MobiGoal application waste time on waiting for a response from an unreachable user and thus influences its reliability. The usability concern is highly related to the UI of MobiGoal. Most users thought that the information shown in different views was useful, but the way how the information was presented could be more friendly. For example, the goal/task states shown in the **Progress** view are not intuitive for normal users to understand.

In the group interview, the users were asked two questions: 1) What is the most attractive feature of MobiGoal? 2) What is the improvement of MobiGoal that you expect most?

When being asked the most attractive feature of MobiGoal, all the users gave their answers actively. For example, User 1 mentioned[1]:

> *MobiGoal is absolutely a welfare for lazy guys like me. There is no need for me to think about applying which apps and contacting which friends for help to accomplish what I want. Especially it makes full advantage of the cooperation from my friends, making it easier for us to help each other.*

User 6 gave a similar comment:

1. Note that this quotation and the following ones are translated from the users' initial answers in Chinese.

TABLE 4
Execution Process of User Goals

User	Execution Process
User 1	$G1 \rightarrow \mathbf{T1_M} \rightarrow G2 \rightarrow G4 \rightarrow G5 \rightarrow \mathbf{T5_I} \rightarrow \mathbb{T}6_{\mathbb{I}} \rightarrow \mathbb{G}5 \rightarrow G6 \rightarrow \mathbb{T}8_{\mathbb{I}} \rightarrow \mathbb{G}6 \rightarrow \mathbb{G}4 \rightarrow G3 \rightarrow \mathbf{T2_I} \rightarrow \mathbf{T3_I} \rightarrow \mathbf{T4_D} \rightarrow \mathbf{G3} \rightarrow \mathbf{G2} \rightarrow \mathbf{G1}$
User 2	$G1 \rightarrow \mathbf{T1_M} \rightarrow G2 \rightarrow G3 \rightarrow \mathbb{T}2_{\mathbb{I}} \rightarrow \mathbb{T}2_{\mathbb{D}} \rightarrow \mathbb{T}2_{\mathbb{D}} \rightarrow \mathbb{T}2_{\mathbb{D}} \rightarrow \mathbb{T}2_{\mathbb{D}} \rightarrow \mathbb{G}3 \rightarrow G4 \rightarrow G5 \rightarrow \mathbf{T5_I} \rightarrow \mathbb{T}6_{\mathbb{I}} \rightarrow \mathbb{G}5 \rightarrow G6 \rightarrow \mathbf{T8_I} \rightarrow \mathbf{T9_M} \rightarrow \mathbf{T10_D} \rightarrow \mathbf{G6} \rightarrow \mathbf{G4} \rightarrow \mathbf{G2} \rightarrow \mathbf{G1}$
User 3	$G1 \rightarrow \mathbf{T1_M} \rightarrow G2 \rightarrow G3 \rightarrow \mathbb{T}2_{\mathbb{I}} \rightarrow \mathbf{T2_D} \rightarrow \mathbf{T3_I} \rightarrow \mathbf{T4_D} \rightarrow \mathbf{G3} \rightarrow \mathbf{G2} \rightarrow \mathbf{G1}$
User 4	$G1 \rightarrow \mathbf{T1_M} \rightarrow G2 \rightarrow G3 \rightarrow \mathbb{T}2_{\mathbb{I}} \rightarrow \mathbb{T}2_{\mathbb{D}} \rightarrow \mathbb{T}2_{\mathbb{D}} \rightarrow \mathbb{T}2_{\mathbb{D}} \rightarrow \mathbb{T}2_{\mathbb{D}} \rightarrow \mathbb{G}3 \rightarrow G4 \rightarrow G5 \rightarrow \mathbb{T}5_{\mathbb{I}} \rightarrow \mathbb{T}5_{\mathbb{D}} \rightarrow \mathbb{T}5_{\mathbb{D}} \rightarrow \mathbb{G}5 \rightarrow G6 \rightarrow \mathbf{T8_I} \rightarrow \mathbf{T9_M} \rightarrow \mathbf{T10_D} \rightarrow \mathbf{G6} \rightarrow \mathbf{G4} \rightarrow \mathbf{G2} \rightarrow \mathbf{G1}$
User 5	$G1 \rightarrow \mathbf{T1_M} \rightarrow G2 \rightarrow G3 \rightarrow \mathbf{T2_I} \rightarrow \mathbf{T3_I} \rightarrow \mathbf{T4_D} \rightarrow \mathbf{G3} \rightarrow \mathbf{G2} \rightarrow \mathbf{G1}$
User 6	$G1 \rightarrow \mathbf{T1_M} \rightarrow G2 \rightarrow G3 \rightarrow \mathbb{T}2_{\mathbb{I}} \rightarrow \mathbb{T}2_{\mathbb{D}} \rightarrow \mathbb{T}2_{\mathbb{D}} \rightarrow \mathbb{T}2_{\mathbb{D}} \rightarrow \mathbb{T}2_{\mathbb{D}} \rightarrow \mathbb{G}3 \rightarrow G4 \rightarrow G5 \rightarrow \mathbb{T}5_{\mathbb{I}} \rightarrow \mathbb{T}5_{\mathbb{D}} \rightarrow \mathbb{T}5_{\mathbb{D}} \rightarrow \mathbb{G}5 \rightarrow G6 \rightarrow \mathbb{T}8_{\mathbb{I}} \rightarrow \mathbb{G}6 \rightarrow \mathbb{G}4 \rightarrow \mathbb{G}2 \rightarrow \mathbb{G}1$
User 7	$G1 \rightarrow \mathbf{T1_M} \rightarrow G2 \rightarrow G4 \rightarrow G6 \rightarrow \mathbb{T}8_{\mathbb{I}} \rightarrow \mathbb{G}6 \rightarrow G5 \rightarrow \mathbf{T5_I} \rightarrow \mathbf{T6_I} \rightarrow \mathbf{T7_D} \rightarrow \mathbf{G5} \rightarrow \mathbf{G4} \rightarrow \mathbf{G2} \rightarrow \mathbf{G1}$

MobiGoal just seems like a powerful and smart agent that integrates software services and social cooperation traditionally provided by different apps, such as searching for specific information, placing orders and social communication with my friends. What's more, it can autonomously and methodically manage these resources and adapt the strategies for the achievement of my personal goals.

In addition, User 3 made a comment from another point of view:

I like the way in which MobiGoal selects candidates in task delegation. MobiGoal provides a convenient way for soliciting help by automatic candidate selection and interaction. If without MobiGoal, I have to repeatedly contact my friends through phone calls or instant messages to confirm their locations and send them task instructions. After that, I still have to wait for their phone calls or instant messages to get the response.

When being asked the expected improvement of MobiGoal, besides the concerns on its usability and sensitivity to network condition, most users expressed their expectation on better customizability. For example, User 7 mentioned that:

I hope I can not only customize the priority orders of subgoals/tasks, but also edit the goal model itself such as adding or deleting goals/tasks. For example, besides the strategies for book acquisition currently supported by the MobiGoal application, I want to add a new strategy of getting a book by borrowing it from my friend.

In summary, most of the users were satisfied with their experience of MobiGoal and expressed their willingness to use MobiGoal in their everyday life. The advantages of MobiGoal, from the users' point of view, are mainly on its autonomous integration of different kinds of resources and adaptation of execution strategies for the achievement of personal goals. Expected improvements of MobiGoal are mainly on better reliability under unstable wireless network, more friendly user interfaces, and more customizable goal models. The current implementation of MobiGoal is a research prototype and suffers all the quality deficiencies of prototypes. We will continuously improve the usability, reliability as well as the cooperation mechanisms of MobiGoal to make it more practical in real life.

6.4 RQ3: Development Effort

Table 5 shows the development effort of the application with the two different development methods, i.e., MobiGoal-based development and ad-hoc development. The four columns respectively show the method used for development, the development content, the development effort by lines of code (LOC), and development time.

The MobiGoal-based development consisted of two phases. First, the developer spent two hours on learning MobiGoal and the syntax of goal model description since it was her first time developing with MobiGoal. After that, she wrote 108 lines of XML code to define the goal model, which took her one hour.

The ad-hoc development included business logic development and UI development. In the development, the developer wrote 3,062 lines of Java code, which took her 13 hours. Among them, 665 lines of code were developed for the business logic of the application, such as the user customization and the scheduling and adaptation of involved tasks. The other 2,397 lines of code were developed for the Android UI for user interaction.

From the comparison, it can be seen that the developer saved 77% time and 96% lines of code while developing applications for specific personal goals with MobiGoal. MobiGoal provides an infrastructure that implements goal- and task-level adaptation logic and interaction logic among different clients. On the other hand, MobiGoal can generate user interfaces for manual work and social delegation based on task description. Therefore, with MobiGoal the developer can focus only on goal model definition. In contrast, in ad-hoc development, the developer implemented the adaptation logic (e.g., by using branch statements or design patterns) and user interfaces by herself.

Note that among the three hours spent on MobiGoal development, two hours were used for the developer to learn MobiGoal development, which can be further saved when the developer is familiar with MobiGoal. Moreover, currently the developer needs to manually write XML code

TABLE 5
Development Effort of Different Methods

Method	Content	LOC	Time(h)
MobiGoal	learning development based on MobiGoal	N/A	2
	goal model development in XML code	108	1
ad-hoc	business logic development in Java code	665	13
	UI development in Java code	2397	

to define a goal model. Therefore, the effort spent on goal model definition can be further saved by providing a visual goal model editor.

In summary, MobiGoal can save 77% time and 96% lines of code for developing an application for a specific personal goal. Moreover, the effort can be further saved if the developer is familiar with MobiGoal framework and given a visual goal model editor.

6.5 Threats to Validity

The threats to the validity of our empirical study are mainly on its external validity.

One major threat to the external validity is the representativeness of the experimental settings, including the selection of users and developers and the simulated campus environment. The students who play various roles in our study may not be representative in practice. For example, the two pedestrians in our study were always ready to help accomplish delegated tasks, while pedestrians in practice are not always willing to help. Application environments in practice are usually much more complex than the simulated campus environment. For example, the routes to the target location of a task and the traffic conditions may be more complex and there may not be enough persons near the target location. The above factors may affect the success rate of adaptation and goal achievement. Although our focus is mainly on an adaptive framework that can combine service invocation, social cooperation, and manual work for user goals, it is worthwhile to integrate more sophisticated cooperator selection and cooperation planning mechanism to improve the success rate of adaptation in more complex situations.

Another major threat to the external validity lies in the fact that MobiGoal and the implementation were only evaluated with a book acquisition application. Our findings may not be applicable to more sophisticated applications with complex business processes and hundreds of tasks. This problem can be alleviated by defining more sophisticated behavior annotations and developing corresponding interpreter. On the other hand, goal models for personal goals in everyday life are usually not so complex, so we think our findings can be generalized in most of the personal applications in practice given proper service stubs and social network platforms.

7 Discussion

Computing devices, human society, and physical objects will soon be seamlessly integrated together, and as an emerging software paradigm for Internet computing, Internetware will orchestrate information, processes, decisions, and interactions in this Internet environment [19]. MobiGoal is targeted at mobile applications running on smartphones and leverages the advantage of mobile applications on the converging of computing devices, human society, and physical objects. Mobile applications can easily invoke software services via mobile network, communicate and cooperate with friends via mobile social network, interact with the users via user interfaces, and access physical objects via embedded sensors. MobiGoal provides a goal-oriented framework for developing Internetware that can orchestrate software services, social cooperation, and manual work for user goals.

The benefits of MobiGoal can be understood from two perspectives. From the perspective of mobile users, MobiGoal actively supports customization and adaptation for mobile applications by integrating into one application different solutions for one goal. For example, library book search is usually provided by a library assistant application; second-hand book search is usually provided by a C2C (consumer-to-consumer) commerce application; location-based friend discovery and task delegation are usually supported by a social network application. From the perspective of application developers, MobiGoal provides the concepts, tools and runtime infrastructure for developing adaptive and customizable mobile applications. With MobiGoal, developers can focus on the definition of alternative solutions for a specific goal in the form of behavioral goal model and the development/integration of software services for automatic task execution.

MobiGoal proposed in this paper is still a prototype in early stage, and it faces many other challenges. MobiGoal obtains human-related events and integrates manual work for task execution via screen-based interaction on the client UI. With wearable devices, it is possible for MobiGoal to support more natural and efficient human computer interaction. For example, by recognizing human actions and status, MobiGoal can capture lifecycle events or check state transition conditions of health-related goals from body sensors.

There are privacy issues with the social delegation supported by MobiGoal. For a user asking for task delegation, his/her private information such as address, bought goods may be disclosed to the cooperators. For a user accomplishing tasks for others, his/her position may be disclosed to the platform for cooperator selection. To alleviate these problems, we can adopt anonymity strategy when possible, for example a user can be anonymous when he/she accomplishes a task for others if the task does not require face-to-face interaction. On the other hand, we can allow users to limit the scope of cooperator selection for different kinds of tasks. For example, a user can set a limitation that only users in his/her group of classmates can be selected for the task of getting book.

Currently, MobiGoal supports social delegation by utility-based user selection and agent-based communication. When used in practice, social issues such as motivation,

human skill, quality control [5] need to be considered. To motivate users to participate social cooperation, some kind of incentive mechanisms need to be introduced based on a social platform. It can be reward points that promote the exchange of value between the consumers and providers. A user earns points for accomplishing tasks for others and consumes points for tasks delegated to others. Moreover, the cooperator selection mechanism can be improved by considering more factors such as the skill and expertise of candidate users based on their profiles and past task accomplishing records. Quality control verifies the accomplishment of goals/tasks and especially delegated tasks. Currently, MobiGoal relies on the confirmation of cooperators for the accomplishment of delegated tasks. However, some cooperators may misinform the success of delegated tasks or more seriously cheat the user. The verification can be done in different ways. The verification of some critical tasks can be regarded as a separate task. For example, the verification of a book reservation can be done by invoking a query service provided by the library or asking another user to confirm it from the library.

MobiGoal depends on alternative solutions expressed in predefined goal models to achieve user goals. In practice, solutions for a common user goal can be continuously evolving. For example, some users may find a new website that provides electronic book search and download services. This can result in an alternative subgoal "Acquired by Downloading Ebook" being added to the goal model for acquiring book. To support evolving requirements with collective intelligence, it is worthwhile to extend MobiGoal with end-user programming capabilities to allow end users to contribute to the evolution of shared goal models.

8 Related Work

Many studies have been focused on dynamic service composition and adaptation towards user goals. Our previous work [11] proposed a requirements-driven self-optimization approach of composite services using feedback control. It maps reconfigurations of a goal model to adaptation decisions of a composite service at both the business process level and service level. Liu et al. [20] proposed a data-driven composition approach for service-oriented situational Web applications. It supports intuitive expression of user's desired composition goals by simple queries and provides a planning technique to exploit composition solutions which can constitute the desired goals. These approaches do not consider the integration of social cooperation with service invocation for user goals. Neither do they support runtime lifecycle management of goal instances.

Goal models have been used in requirements-driven self-adaptation to reason about the fulfilment of goals and make adaptation decisions at runtime. Baresi et al. [7] proposed an approach that introduces fuzzy and adaptive goals to embed adaptation countermeasures and fosters self-adaptation by considering goals as live, runtime entities. Wang et al. [8] proposed a goal-oriented framework that exploits high variability in a software system to deliver self-repair capabilities through reconfiguration. Dalpiaz et al. [9] proposed a self-reconfiguration architecture by enriching goal models with context-dependent goal decompositions, goal commitments with time limits and domain assumptions. Peng et al. [10] proposed a self-optimization approach that uses preference-based goal reasoning to reason about optimal goal configurations. Qian et al. [21] proposed an approach that combines goal reasoning and case-based reasoning for self-adaptation. Some researches use goal-oriented frameworks to support the development of adaptive and customizable systems. Lapouchnian et al. [22] proposed an approach for designing self-adaptive systems based on goal models. Liaskos et al. [23] proposed a goal-oriented approach for behavioral customization of information systems. MobiGoal applies goal-oriented adaptation and customization in mobile applications and provides an agent-based framework that integrates automatic task execution, manual work, and social delegation for user goals.

Dalpiaz et al. [14] formalized a participant's strategy for particular goals in terms of goals, plans and commitments and proposed a conceptual model and framework for adaptive agents in open systems based on the notion. Our previous work [15] proposed a goal state machine to model the runtime lifecycle of goal instances, which supports social delegation and self-repairing. MobiGoal's runtime goal model extends it with top-down activation and execution scheduling of goal and task instances. Moreover, MobiGoal provides an agent-based implementation framework for developing mobile applications for user goals.

In mobile computing area, mobile sensing has been widely used to allow users to share their personal information and consume emerging services for personal needs [1]. Researchers developed approaches and mobile applications that use mobile sensing, for example, to estimate individual exposure to environmental pollution [24], or predict bus arrival time [25]. These approaches use shared personal sensing data to satisfy information needs of users but cannot support users to achieve high-level goals.

Several research efforts have focused on support for the development of self-adaptive mobile applications. Zachariadis et al. [26] proposed a lightweight component meta-model with logical mobility primitives for the construction of adaptive mobile applications supporting redistribution of components. Rouvoy et al. [27] proposed a middleware platform which supports component-based adaptation for mobile applications. These approaches focus on component model and middleware platform that support component-based adaptation in mobile computing environment and do not consider user goals and requirements-driven adaptation.

9 Conclusion

To support users to achieve high-level goals, mobile applications need to adaptively switch among alternative solutions and combine automatic execution, manual work, and social delegation for task accomplishment. Ad-hoc application development for each individual goal is inefficient and hard to support personalized requirements and future changes.

In this paper, we have proposed an agent-based and goal-oriented framework called MobiGoal, which provides a systematic and efficient way for developing such mobile applications. MobiGoal's runtime goal model enables adaptive goal activation and execution based on embedded behavioral information and goal state machine for the lifecycle

management of each goal instance. On the other hand, MobiGoal provides an agent-based framework that integrates different task execution modes (automatic service invocation, manual work, and social delegation) into a unified goal lifecycle management. With MobiGoal, developers can develop a mobile application for a high-level goal by defining a behavioral goal model for it and developing specific software services as required. After obtaining and loading a goal model into mobile client, a user can customize it into a mobile application for specific goals. We have developed an implementation of MobiGoal for Android platform and evaluated the effectiveness of MobiGoal with a preliminary empirical study.

Our future work will improve the user interaction and social cooperation of MobiGoal, including UI design, integration of mobile social network platform, incentive mechanism in social delegation. In addition, we plan to conduct further evaluation of MobiGoal, since our preliminary evaluation was carried with only few participants. Finally, we intend to study the evolution of MobiGoal applications by collecting and learning from feedback from mobile users.

ACKNOWLEDGMENTS

This work is supported by National High Technology Development 863 Program of China under Grant No. 2015AA01A203.

REFERENCES

[1] N. D. Lane, E. Miluzzo, H. Lu, D. Peebles, T. Choudhury, and A. T. Campbell, "A survey of mobile phone sensing," *Commun. Mag.*, vol. 48, no. 9, pp. 140–150, 2010.

[2] S. Dhar and U. Varshney, "Challenges and business models for mobile location-based services and advertising," *Commun. ACM*, vol. 54, no. 5, pp. 121–128, 2011.

[3] J. Chon and H. Cha, "Lifemap: A smartphone-based context provider for location-based services," *IEEE Pervas. Comput.*, vol. 10, no. 2, pp. 58–67, 2011.

[4] Y. Zheng, Y. Chen, X. Xie, and W.-Y. Ma, "Geolife2.0: A location-based social networking service," in *Proc. 10th Int'l Conf. Mobile Data Management*, 2009, pp. 357–358.

[5] A. J. Quinn and B. B. Bederson, "Human computation: A survey and taxonomy of a growing field," in *Proc. SIGCHI Conf. Human Factors in Computing Systems*, 2011, pp. 1403–1412.

[6] P. Sawyer, N. Bencomo, J. Whittle, E. Letier, and A. Finkelstein, "Requirements-aware systems: A research agenda for RE for self-adaptive systems," in *Proc. 18th IEEE Int'l Requirements Eng. Conf.*, 2010, pp. 95–103.

[7] L. Baresi, L. Pasquale, and P. Spoletini, "Fuzzy goals for requirements-driven adaptation," in *Proc. 18th IEEE Int'l Requirements Eng. Conf.*, 2010, pp. 125–134.

[8] Y. Wang and J. Mylopoulos, "Self-repair through reconfiguration: A requirements engineering approach," in *Proc. 24th IEEE/ACM Int'l Conf. Automated Software Eng.*, 2009, pp. 257–268.

[9] F. Dalpiaz, P. Giorgini, and J. Mylopoulos, "An architecture for requirements-driven self-reconfiguration," in *Proc. 21st Int'l Conf. Advanced Information Systems Eng.*, 2009, pp. 246–260.

[10] X. Peng, B. Chen, Y. Yu, and W. Zhao, "Self-tuning of software systems through dynamic quality tradeoff and value-based feedback control loop," *J. Syst. Softw.*, vol. 85, no. 12, pp. 2707–2719, 2012.

[11] B. Chen, X. Peng, Y. Yu, and W. Zhao, "Requirements-driven self-optimization of composite services using feedback control," *IEEE T. Services Computing*, vol. 8, no. 1, pp. 107–120, 2015.

[12] N. Bencomo, J. Whittle, P. Sawyer, A. Finkelstein, and E. Letier, "Requirements reflection: Requirements as runtime entities," in *Proc. 32nd IEEE/ACM Int'l Conf. Software Eng.*, 2010, pp. 199–202.

[13] F. Dalpiaz, A. Borgida, J. Horkoff, and J. Mylopoulos, "Runtime goal models," in *Proc. 7th IEEE Int'l Conf. Research Challenges in Information Science*, 2013, pp. 1–11.

[14] F. Dalpiaz, A. K. Chopra, P. Giorgini, and J. Mylopoulos, "Adaptation in open systems: Giving interaction its rightful place," in *Proc. 29th Int'l Conf. Conceptual Modeling*, 2010, pp. 31–45.

[15] L. Fu, X. Peng, Y. Yu, J. Mylopoulos, and W. Zhao, "Stateful requirements monitoring for self-repairing socio-technical systems," in *Proc. 20th IEEE Int'l Requirements Eng. Conf.*, 2012, pp. 121–130.

[16] X. Peng, Y. Xie, Y. Yu, J. Mylopoulos, and W. Zhao, "Evolving commitments for self-adaptive socio-technical systems," in *Proc. 19th Int'l Conf. Eng. of Complex Computer Systems*, 2014, pp. 98–107.

[17] "JADE," http://jade.tilab.com, 2015.

[18] J. Mylopoulos, L. Chung, and B. Nixon, "Representing and using nonfunctional requirements: A process-oriented approach," *IEEE Transactions on Software Engineering*, vol. 18, no. 6, pp. 483–497, 1992.

[19] H. Mei, G. Huang, and T. Xie, "Internetware: A software paradigm for internet computing," *IEEE Computer*, vol. 45, no. 6, pp. 26–31, 2012.

[20] X. Liu, Y. Ma, G. Huang, J. Zhao, H. Mei, and Y. Liu, "Data-driven composition for service-oriented situational web applications," *IEEE T. Services Computing*, vol. 8, no. 1, pp. 2–16, 2015.

[21] W. Qian, X. Peng, B. Chen, J. Mylopoulos, H. Wang, and W. Zhao, "Rationalism with a dose of empiricism: Case-based reasoning for requirements-driven self-adaptation," in *Proc. 22nd IEEE Int'l Requirements Eng. Conf.*, 2014, pp. 113–122.

[22] A. Lapouchnian, Y. Yu, S. Liaskos, and J. Mylopoulos, "Requirements-driven design of autonomic application software," in *Proc. Conf. Center for Advanced Studies on Collaborative Research*, 2006, pp. 80–94.

[23] S. Liaskos, M. Litoiu, M. D. Jungblut, and J. Mylopoulos, "Goal-based behavioral customization of information systems," in *Proc. 23rd Int'l Conf. Advanced Information Systems Eng.*, 2011, pp. 77–92.

[24] A. B. Waluyo, D. Taniar, B. Srinivasan, and W. Rahayu, "Mobile query services in a participatory embedded sensing environment," *ACM Trans. Embed. Comput. Syst.*, vol. 12, no. 2, pp. 31:1–31:24, 2013.

[25] P. Zhou, Y. Zheng, and M. Li, "How long to wait?: predicting bus arrival time with mobile phone based participatory sensing," in *Proc. 10th Int'l Conf. Mobile Systems, Applications, and Services*, 2012, pp. 379–392.

[26] S. Zachariadis, C. Mascolo, and W. Emmerich, "The SATIN component system-a metamodel for engineering adaptable mobile systems," *IEEE Trans. Softw. Eng.*, vol. 32, no. 11, pp. 910–927, 2006.

[27] R. Rouvoy, M. Beauvois, L. Lozano, J. Lorenzo, and F. Eliassen, "Music: an autonomous platform supporting self-adaptive mobile applications," in *Proc. 1st Workshop on Mobile Middleware: embracing the personal communication device*, 2008, pp. 6:1–6:6.