Situ: A Situation-Theoretic Approach to Context-Aware Service Evolution

Carl K. Chang, *Fellow*, *IEEE*, Hsin-yi Jiang, Hua Ming, and Katsunori Oyama

Abstract—Evolvability is essential for computer systems to adapt to the dynamic and changing requirements in response to instant or delayed feedback from a service environment that nowadays is becoming more and more context aware; however, current context-aware service-centric models largely lack the capability to continuously explore human intentions that often drive system evolution. To support service requirements analysis of real-world applications for services computing, this paper presents a situation-theoretic approach to human-intention-driven service evolution in context-aware service environments. In this study, we give *situation* a definition that is rich in semantics and useful for modeling and reasoning human intentions, whereas the definition of *intention* is based on the observations of situations. A novel computational framework is described that allows us to model and infer human intentions by detecting the desires of an individual as well as capturing the corresponding context values through observations. An inference process based on Hidden Markov Model makes instant definition of individualized services at runtime possible, and significantly, shortens service evolution cycle. We illustrate the possible applications of this framework through a smart home example aimed at supporting independent living of elderly people.

Index Terms—Context aware, hidden Markov chain, human intention, requirements, runtime, service, situation theoretic, smart home, software evolution.

1 Introduction

Evolvability is essential for computer systems to adapt to the dynamic and changing requirements in response to instant or delayed feedback from a service environment that nowadays is becoming more and more context aware [1], [2]; however, current Context-Aware (CA) service-centric models largely lack the capability to continuously explore human intentions that often drive system evolution [3]. For system designers, modeling and capturing a large set of context parameters are important for modern-day, context-aware, and often sensor-laden computer applications [4]. As such, system engineering tasks are increasingly complex and highly challenges in view of the unstable, noisy, and sometimes harsh environments, where sensors constitute critical elements of CA systems [2]. Meanwhile, requirements engineers working on requirements solicitation are often geared toward goal specification [5], while goal is only a part of the human mental state. Traditionally, human mental states have been extensively studied in the realm of belief, desire, and intention (BDI) [6]. Our research framework also originated from an initial study on the BDI model [7]; later, our framework refines definitions of *situations* and *intentions* in order to adapt to sophisticated computational needs.

We believe that our framework is particularly useful for computer-based applications that are driven by *services*. From the viewpoint of *services computing*, we note that workplace protocols, life styles, healthcare needs, and even laws rapidly evolve day by day in the modern world. Such trends also demand services to evolve rapidly. We use the term *service evolution* to highlight the nature of computer-based applications to prevail in the 21st century from end users' perspective, and *software evolution* to emphasize the mechanism to evolve computer programs as enabler of service evolution from the developer's perspective. Our research framework is founded on the study of software evolution mechanisms to meet the service evolution requirements drawn from the applications world.

It must be realized that services rendered to humans will not completely meet individual needs in all facets and at all times. Since humans are very complex beings, services must evolve to deal with such human complexities. Therefore, the ever-evolving and sudden emerging nature of human mental states tend to dictate the need for service evolution [8]. On the other hand, it has also been noted that environmental changes for real-world systems have become increasingly unpredictable [9]. Today, services computing researchers often advocate that at the design stage, requirements engineers, system designers, and other stakeholders should specify situations as collections of contexts and services to invoke [10]. System and software requirements, however, may not cover all the possible situations. As a result, systems often fail to effectively evolve to accommodate incomplete and dynamically changing requirements because of their inability to acquire context values or constantly changing human intentions. This study only focuses on the human-intention aspects of service evolution by detecting human intention changes. Modeling and managing failures due to harsh, unstable, and unsafe environments, where context values cannot be easily collected or derived, can be found in other studies (e.g., [2]).

The novelty of this study lies with our definition of *situation* that is semantically richer than the prevailing definitions [11], [10] for CA services environment. We suggest that situations perceived by a human depend on the

• *The authors are with the Department of Computer Science, Iowa State University, 226 Atanasoff Hall, Ames, IA 50011.*
E-mail: {chang, hsinyij, hming, oyama}@cs.iastate.edu.

Manuscript received 17 Feb. 2009; revised 18 June 2009; accepted 8 July 2009; published online 28 July 2009.
For information on obtaining reprints of this article, please send e-mail to: tsc@computer.org, and reference IEEECS Log Number TSCSI-2009-02-0029.
Digital Object Identifier no. 10.1109/TSC.2009.21.

human internal mental state, and the actions performed can be regarded as an external reflection. Note that context value changes are side effects of human actions that are externally observable. This new definition for situation affords us a high-level computational playground. Also note that our definition of situations is influenced by the meaning of situation defined in Situation Theory [12], although ours is directly tied with human mental state exploitation and peculiar to service evolution. Further, due to the semantically rich definition of situation, we also present a unique definition of *intention* that enables us to develop service specification paralleling intention specification. Such intention and service specifications can help complete services requirements analysis to allow developers to perform service definition, packaging, and possibly dynamic composition. Our basic assumption is that an original set of system parameters (goal tree, situations, intentions, behavioral predictive probabilities) has been made available during deployment time in a repository. However, if it is not the case, we briefly present a partial solution in Section 4.2.

In general, studies on intention treated it as a plan or an execution path [13]. This paper argues that intention can be inferred from situations and constructed from a window of observations to formulate a temporal sequence of situations with the aim to achieve a goal. Remember that humans are open systems; they can learn and often change past desires. Changing desires are invisible and not directly observable during service runtime but often lead to a new goal. That is, with current models of software development, predefined system requirements may eventually become temporarily obsolete or contrary to an intention. Moreover, the changed intention often results in a new goal for the user, requiring the modification of existing features or new pathways to engage new development, such as creating a new functionality. Oftentimes, such a cycle of observation, modification, development, and deployment takes a long time to complete, say several months, by following, the current common software life-cycle processes. Researchers are now facing the steep challenge of shortening such undesirably long evolution cycles, perhaps in terms of a few weeks to a few days, or even shorter so that critical and timely *service individualization* can become a reality. We propose to accomplish this ambition by observing human action and predicting transitions of human mental states. We identify Hidden Markov Model (HMM) with Dirichlet Process [14] as a suitable mathematical model to address this runtime prediction issue. Equipped with such a prediction mechanism, a new runtime service evolution framework, *Situ*, comes to life to support rapid and iterative service requirements analysis of real-world systems.

This paper is organized as follows: Section 2 briefly reviews the definitions and concepts of software evolvability, human intention, and context-aware and situation-aware research. Section 3 describes a new runtime service evolution framework centered around a particular situation theory, explains our strategy to exploit service feedback from observations, and presents how intentions can be monitored and inferred via the *Situ* framework. Section 4 describes an observation algorithm for deriving human mental states using HMM. A smart home example is employed for illustration. Finally, Section 5 enumerates research challenges and Section 6 concludes the paper with some speculations.

2 Related Work

In this section, we review the literature on software evolution, context-aware and situation-aware computing, and contrast those views against ours pertaining to software evolvability and human intentions.

2.1 Software Evolvability

Belady and Lehman [15] first applied the term *evolution* to software. Since then, most researchers have used that term to refer to various types of changes in software systems. In their approach, evolvability entailed the effort to measure complexity, size, cost, and maintenance dimensions of software changes, based on a study of the OS/360 operating system. In that study, Lehman et al. identified the first law of software evolution (*continuing change*) [16], which states: *An evolvable system must be continually adapted otherwise it becomes progressively less satisfactory.* Furthermore, software evolution also leads to high complexity as the second law of software evolution (*increasing complexity*) dictates: *As a system evolves, its complexity increases unless work is done to maintain or reduce it.* In most cases, increasing complexity seriously affects system quality as it acquires new functionality [17].

From the viewpoint of business environments, evolvability is the capacity of a system to adapt to changing requirements throughout its life cycle without compromising architectural integrity. An evolvable system must meet the new needs of a customer in the most cost-effective manner. In a broader view, environmental changes include factors pertaining to system environment, requirements, and implementation technologies [18]. For service-oriented pervasive environments, constantly changing intentions of the customer effectively drive requirements evolution and incremental development [19]. Therefore, human intentions play an important role in triggering system feedback for deriving new and preferred services or features in the targeted environment. If they never adapt to environmental changes, all real-world systems may eventually decay into disservice or nonservice.

2.2 Contexts and Situations

Environmental changes are characterized as *contexts* in pervasive computing. Context is often referred to as "any information that can be used to characterize the situation of an entity" [4] for providing relevant information and/or services to the user. The state of the art for CA research tends to center around designing context models (the most popular ones are logic-based models or ontology-based models [20]), and developing various context reasoning and monitoring methods. Many existing CA approaches synthesize information about location, object, physiological readings of people, etc., for real-world applications. For example, in pervasive computing, location-based services provide personalized functions based on the location gathered from context information [21].

Most context and situation models have limited capability to explore human intentions for evolvability. Marschall [22] pointed out that software developers tend to lose interest and get frustrated if they are not able to make a direct connection between their contribution to their customers and the observable outcomes. This disconnection leads to low product quality. In his study, the results of the customer's intention were not adequately addressed. It is imperative that system designers begin to seriously investigate human-intention-driven development and evolution methods. The end goal is to move beyond the prevailing personalization (largely only parameterized or template driven) to afford real individualization, which requires rich information of situations to perceive a user's demands for service.

Historically speaking, situation was originally studied in order to investigate natural language semantics. Later, an information-based theory called *situation theory* was developed. Traditionally, the theory seeks to understand linguistic utterances in terms of information conveyed. Barwise and Perry defined situation as parts of the world, which can clearly be recognized in common sense and human language [12]. Based on their definition, events and episodes are situations in time. In the 1990s, it was shown that situation theory could be applied to analyze language with actions. Although the term *situation* is the same, researchers have different perspectives on defining it, especially when the objectives are different. For instance, some researchers defined situation as "histories" (i.e., finite sequences of primitive actions [23]—thus, corporate memory is maintained), or a set of contexts in a system over a period of time [10] that evolves. Situation can be also reasoned by aggregating a set of predicates that explicitly represents information about sensory data [11]. Sequencing such predicates can form the history of a finite number of instances in the first-in first-out order. For our purpose on understanding human intention, situations should be periodically captured from context with respect to human mental states. In this paper, we define situation in a human-centric fashion (see Section 3.3).

2.3 Human Intention

Although human intention has been well investigated by many researchers in Philosophy, Cognitive Science [24], and Artificial Intelligence [25], [26], the definition of *intention* is still varied and sometimes controversial. Definitions often attempt to deal with how human mental states are viewed, and purposeful definitions vary between disciplinary areas.

Bratman, a philosopher, describes intention as one of the BDI mental states, which motivates action [6]. AI researchers have used his BDI model and consequently have developed agent-planning techniques; however, in most AI-based research, intention is only considered as an execution plan of an agent [13]. Currently, pure BDI agent models (e.g., [27], [28], [29]) focus on finding an optimal plan to reach the declarative goal under a given environment. However, the plan in BDI models is not enough for observing human intention. For example, locally heterogeneous plans often mean an identical intention for an individual system user. A rigorous computational model to precisely capture human intention is critically needed in our research.

In response to Bratman's theory, Scheer creates his opposing definition to avoid the problems associated with modeling mental states in the BDI fashion, defining intention as simply a course of action that one has adopted [30]. In fact, both definitions seem to agree that intentions can be analyzed through observation of an individual's actions. Recently, observation of human intention has been studied in sensor-based approaches [31], [32]. Note that there has been an emerging research focus on temporal context-awareness, where user-centered situations, such as location, can be taken into account for computation to support improved life style [33]. For analyzing the intention change that triggers software evolution, we define intention as temporal sequence of situations to achieve a goal (see Section 3.3 for more detail); any intention is supposed to work as a constraint for satisfying the goal, which is specified as requirements.

2.4 Human-Intention-Based Software and Requirements Engineering

Both human intentions and goals play important roles in User-Centered Design (UCD) to address user concerns. Norman formulated that interaction between humans and computers consists of seven cyclic stages [34, Norman's action theory]. In his theory, a goal is realized by intentions. Each intention is considered as a mental thread, which performs a set of required or requested tasks. In addition, a goal is also considered as a desired end state to be achieved by the user in Norman's action theory, rather than a state to be reached after successful execution of a task in the traditional task analysis typically seen in other's work [35].

In Requirements Engineering (RE), the notion of *goal* is widely accepted to analyze requirements [5]. Goals are optative statements and similar to human desires. Some goals are decomposable and consistent with each other in the *goal tree* based on the AND/OR relationships. The interrelationships between goals and intentions have been studied by researchers to realize a context-aware service [36]. The generally accepted notion is that every target of an intention can be represented as a goal, where they are relational and complementary concepts.

Usually goals are explicitly stated by stakeholders in the preliminary material available to requirements engineers. However, goals can be initially implicit for system designers and even users themselves. Because some goals, especially nonfunctional ones, are partly tacit knowledge, goal elicitation and resolution are complex issues [37]. Another important factor is that the everchanging situations of a modern-day service environment (say, for Web services) constantly demand service evolution, for users' satisfaction. How to capture and predict new goals during runtime becomes an eminent research challenge; a robust prediction-based modeling framework is required.

Fortunately, recent research in Bayesian network [38], [39] has been successful in inferring a few types of situations during a certain period of time. In particular, a specific type of dynamic Bayesian network called the HMM [40] suits our research objectives. In a typical HMM, human desires are represented as hidden states and contexts are considered as visible variables (see Section 4.1). Nevertheless, in most previous research, intentions derived from

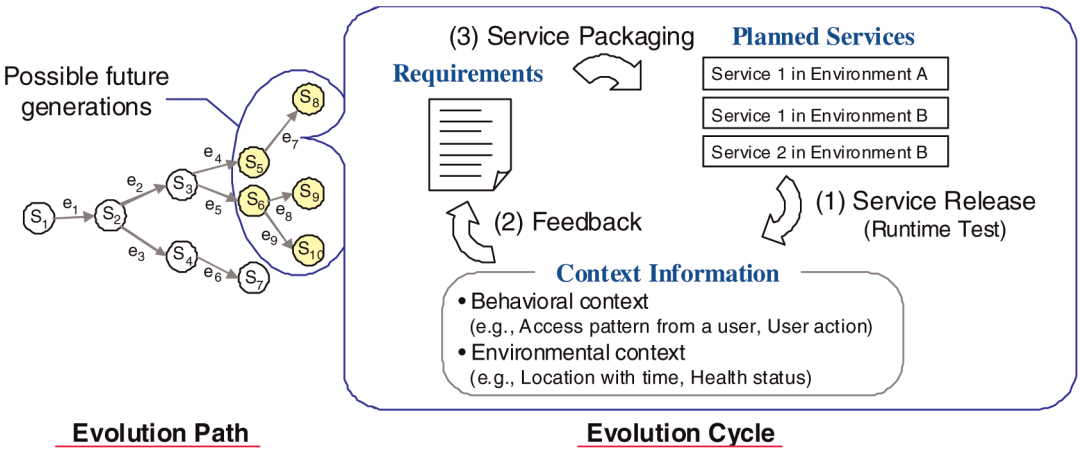

Fig. 1. Service evolution @ runtime.

situations are still not *high-level behavior* [39] (e.g., an intention explains only a temporary goal without the *corporate memory*) and microscopic (e.g., no end-to-end scenario to relate human goal to human behavior based on the observations over a period of time) for software developers to define a wholesome updated service goal.

In our view, a goal is characterized by a set of chosen desires as a *compound desire* and the corresponding intentions, which can be inferred from a group of situations. In general, more than one human intention can achieve a goal. In other words, humans can generally find different ways to achieve a goal. For monitoring intention changes, we define situation and intention in a human-centric fashion (see Section 3.3). Compared to other research, our contributions are 1) formulating both situations and intentions in precise terms and 2) modeling a runtime service evolution process in a computational manner. This service evolution process called *Service Evolution @ Runtime* includes the activities to observe intentions based on the temporal sequence of situations with respect to a certain set of desire states predicted by HMM. While monitoring intentions, system designers attempt to maintain and update service goals to meet each individual user's evolving desires.

3 Service Evolution @ Runtime

In this study, our requirements analysis technique is intended to instantly predict possible future generations of services and assess the threat of quality loss by observing intentions from contexts. Traditionally, software evolution only observes historical defects and changes in program codes [17], [16], [41]. In contrast, human-intention-driven approach focuses on the runtime service evolution caused by an intention change. We believe that intention changes often indicate potentially valuable, individualized requirements for upgrading a system to a new version or engaging development of new features.

Regarding runtime service evolution, evolution cycle forms a time-branching path (Fig. 1). Software developers closely monitor such an evolution cycle in order to control service quality, while the feedback process provides a method to reflect intention changes in the service environment.

3.1 Evolution Path

A number of release-planning methods have been developed; some are employed for solving the release-planning problems [41] along the evolution path. In our approach, software evolvability refers to the ability of the system to handle human intentions in order to prolong the life of the system usage while facing environmental changes, and safeguard the quality of service in the mean time.

Thus, an evolution path represents the history of software releases. In this path, a collection of tightly coupled intention changes (emerging in the same time period since the last release, and challenged by gradual or sudden situation changes) triggers an evolution cycle. This process may affect the quality of existing software services, while new functionalities may emerge and dictate future goals.

3.2 Evolution Cycle

In an evolution path, each edge denotes an evolution cycle. This feedback process consists of three phases (service release, feedback, and service packaging) described as follows:

1. **Service release.** Each generation of service requires individual tests in different environments. For this purpose, a developing service usually has beta and stable release versions. Some services may have a longer lifetime by satisfying user demands better than others. Since there are unpredictable software defects and intention changes by users, a service has the potential to become undesirable. By analyzing whether the service is adapted to an environment or not, an evolvable system should generate feedback for future service versions.
2. **Feedback.** The feedback phase is decomposed in three subprocesses (Fig. 2). First, through observations, the system captures context variables which are *visible*. An HMM-based mental state model deduces situations and desire states (note that these are *hidden*) of an individual user from given context variables, and proceeds to identify an intention, where "intention" is formally defined through the sequence of captured situations and a derived desire (refer to Definition 3.3.3). When the intention

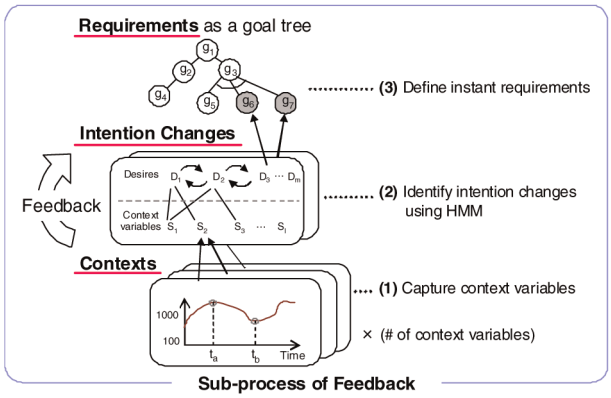

Fig. 2. Subprocesses of Feedback.

shows a change, the system updates its goal tree in reference to the corresponding desire, which represents a possible future goal. This change also means that a system designer, who is monitoring user intention, can identify a changed intention by observing a sequence of situations (e.g., rapid changes of menu selection by the user of a GPS system). Note that the update task is mostly a semiautomatic process, and thus, should be managed by the system designer who is monitoring user intentions. Identification of an intention change is based on the premise that a different sequence of situations is observed when compared to the historical data. What follows is the instant requirements definition process that reflects such intention changes for a new or updated service plan. Moreover, to compare with a subset of the goal tree, constraints of value (e.g., $responsetime \leq 1\ ms$) or sequence (e.g., history of chosen paths to a destination in GPS) should be defined in the instant requirements definition process.

3. **Service packaging.** Evolving a service essentially entails a process of service packaging, which cuts and inserts service plans on a goal tree incrementally in a bottom-up fashion. A service package to be evolved requires an instantly defined goal to be added to the goal tree and the goal-directed intentions as an execution plan. Each instant goal is maintained with a versioning system so that program modifications or developing new services developments can swiftly be engaged when customers need them. A version of service represents a subset of the goal tree. Such individualized requirements can help form a new service package in this manner.

3.3 Intention Monitoring

An intention, mentioned before as a temporal sequence of situations to achieve a goal, plays a proactive role in interpreting human actions, which are the key component connecting desires and specific contexts. In our work, we decompose contexts into two subdomains—environmental contexts and behavioral contexts—in order to find the most likely set of states (i.e., human desires) as shown in Fig. 3.

Environmental Contexts are categorized as lower level contexts, derived from sensing the entire environment surrounded by a user being observed in a service system. This type of contexts (e.g., time, location, and status of an object) is useful in understanding the triggers and effects of human actions. Specifically, some environmental contexts such as foot position and service use pattern are key factors to reason human actions, and the others depict results from interaction with a human action (e.g., state of a door opened by the service user).

Behavioral Contexts, which interact with environmental contexts bidirectionally (i.e., sensing and actuating), are

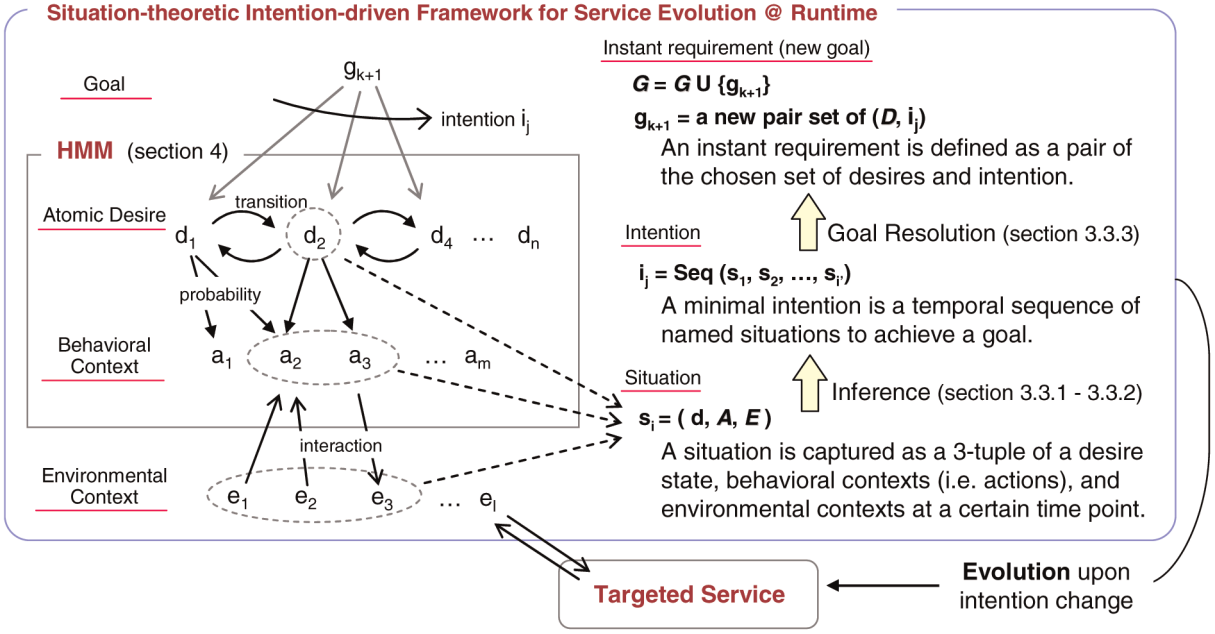

Fig. 3. *Situ*: Framework for service definition and packaging with runtime software evolution.

manifested in human actions. Although some behavioral contexts can also be drawn from raw data (such as wearable sensor data), they are different from environmental contexts. Unlike environmental contexts, behavioral contexts are derived from human behavior, which is considered open and evolvable.

Behavioral changes can oftentimes be reduced to intention changes; the critical issue raised here is how to capture intention changes, through detection of behavioral changes. Given the highly complex human intention mechanism, we propose to address this issue from a multilevel perspective. At the lower level, we apply HMM to capture changes in human mental states through observable-parameters-based, hidden states inference (more detail in Section 4). In this section, we lay the foundation for a higher level formal mechanism to understand human intention change by one of the key components in our framework: the situation. Based on this perspective, we name this research framework *Situ* to reflect its nature as a situation-theoretic intention-driven approach to software evolution.

3.3.1 Inference Process in Situ Framework

We formally define *situation* as follows:

Definition 3.3.1. *A situation at time t, $Situation(t)$, is a triple $\{d, \mathbb{A}, \mathbb{E}\}_t$ in which d is the predicted user's desire, $\mathbb{A}$ is a set of the user's actions to achieve a goal which d corresponds to, and $\mathbb{E}$ is a set of context values with respect to a subset of the context variables at time t.*

In Definition 3.3.1, elements in the set $\mathbb{A}$ can be one or more atomic actions or a composite action (i.e., a sequence of actions). At time t, the sets $\mathbb{A}, \mathbb{E}$ are derived first, and the desire d is predicted by HMM accordingly. In our system, we assume that human desires form a hierarchical structure called a *desire tree*. A compound desire is the root of the desire tree and can be decomposed into smaller subdesires usually represented as an AND/OR graph. Desires in the leaf are atomic and cannot be further decomposed. In both HMM and Definition 3.3.1, only the atomic desires are considered. A research challenge raised here is that in order to predict a desire from a set of actions, the system needs to be able to efficiently convert continuous human actions (e.g., through visual analysis of a video tape) into observed behavioral parameters to feed the HMM. To solve this problem, a preliminary idea on applying known discretization techniques [42], [43] to discern behavioral parameter values is considered. Further investigation is needed to validate this idea.

A desire tree is similar as a goal tree mentioned in Section 2.4. The major difference is that a goal tree is analyzed from system's perspective in order to capture user requirements and a desire tree is analyzed from human's perspective in order to monitor human intentions. For instance, the goals in a goal tree are the requirements for the system to facilitate the user to accomplish his desire. The desires in a desire tree represent a hierarchical set of desires in the user's mind.

Definition 3.3.1 gives a formal definition of general situations. It is not always true that all captured situations are of interest to a particular service system. During the system design process, a set of situations of particular interests needs to be identified, which is called *named situations*. That is, we assume that there exists a namespace for situations. As a general rule of thumb, the values of context variables (both behavioral and environmental contexts) with respect to those named situations are fixed, and some context variables allow tolerance of values that are imprecise, or may fluctuate with regard to specific concerns pertaining to the applications. To claim those named situations, the system designer can provide a specific point or a range, based on the analysis of context data, as the values of context variables; however, the ranges thus determined may overlap with each other. In order to make sure that the system can capture at most one named situation at any point in time, u_{ij} is defined in Definition 3.3.2 for the uniqueness of the named situations with disjoint value sets of the corresponding context variables. The value $Threshold_i$ is to guarantee that the range claimed for c_i for a specific named situation is within the acceptable tolerance due to imprecision or fluctuation of measurement.

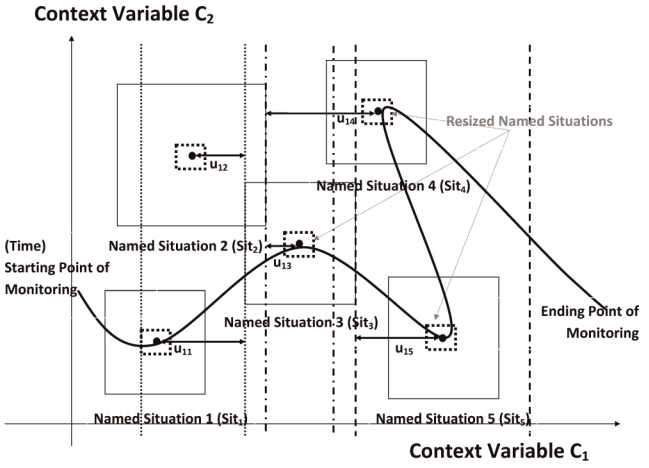

Fig. 4. An example of capturing named situations.

Definition 3.3.2. *Let $v = (v_1, v_2, \ldots, v_n)$ be a vector in which each element is corresponding to context variables $c_1, c_2, \ldots, c_n$ (both behavioral and environmental contexts), respectively, of all the named situations. For each i, let*

$$u_{ij} = \sup\{|\delta_j| \mid \text{for any values of context variables other than } c_i, \text{the range } [m_{ij} - \delta_j, m_{ij} + \delta_j] \text{ contains at most one named situation}\}, \text{for all } j \in I_{sit},$$

where m_{ij} is the fixed (or mean) context value for c_i in named situation j and I_{sit} is the index set of named situations. Each v_i is defined as

$$v_i = \min\left\{Threshold_i, \frac{1}{2}\min\{u_{ik} \mid k \in I_{sit}\}\right\},$$

where $Threshold_i$ is the tolerance for c_i.

Fig. 4 shows an example for Definition 3.3.2 (only two context variables are included for illustration). Combined with predicted desires, situation changes can be captured. In other words, the vector v defined in Definition 3.3.2, with the fixed (or mean) context values of named situations,

forms neighborhoods of named situations. Through those neighborhoods, a temporal sequence of situations can be derived at runtime.

Human intention is intangible; however, it can be inferred by a sequence of observed executions (both human actions and human-system interactions) represented as situations. That is, a sequence of situations generally provides certain information of regularity to characterize an intention. Each intention monitoring can only start when its precondition is met (e.g., a transition to another desire in HMM), and complete when its postcondition is satisfied (i.e., the goal is achieved). An intention is understood as an execution plan of actions and formally defined as follows:

Definition 3.3.3. *With respect to a goal g, an intention is defined as $I = seq(S_1, S_2, \ldots, S_k)$ through which the goal g is achieved, where $k \in N$ represents the length of the sequence. S_1 and S_k are the starting point and ending point of the intention, respectively. $S_i, i \in \{1, \ldots, k\}$, belongs to the named situation set Sit.*

Theoretically, human intentions, including intention changes, should be continuously monitored at every moment while a system is running; however, it is almost impossible for the current state of the art that systems store data continuously at every moment and analyze all of them. With the vector v in Definition 3.3.2, a system monitors an intention by observing sequence of situations. Even with Definition 3.3.3, one major issue still remains; in the real world, although the user purposely follows a procedure to achieve one of his desires, it is possible that some irrelevant actions are included in the middle of the procedure. For instance, to follow the rules for the personal hygiene advised by a doctor, the user wants to reach the goal "take a shower before going to bed," one of the intentions should be $seq\{S_1, S_2, S_3, S_4\}$ in which S_1 (a named situation) corresponds to the action "take out clothes from dryer," S_2 (a named situation) corresponds to the action "prepare shampoo and soap," S_3 (a named situation) corresponds to the action "remove clothes," and S_4 (a named situation) corresponds to the action "adjust water temperature." Each of the named situations has a value set of the corresponding context variables. Assume that the user intends to achieve the goal by executing the above intention. In the middle of S_1 and S_2, the user picks up a phone because the phone rings. Picking up a phone is an irrelevant action with respect to the goal; however, the action "pick up a phone" associated with the environmental context values and the predicted desire form a situation at this moment and it might be captured by the system and match a named situation, say S_1'. After answering the phone call, the user performs the rest of the actions to accomplish the goal. In the system, the monitored intention becomes $seq\{S_1, S_1', S_2, S_3, S_4\}$. In order to discern the relevant situations from the irrelevant ones, *goal-directed situations* are defined as follows:

Definition 3.3.4 (goal-directed situations). *With respect to a goal g, a goal-directed situation is a named situation whose actions are executed to fulfill the compound desire corresponding to the goal g.*

Note that a temporal partial order exists in any set of goal-directed situations. By the same example, it does not matter whether S_1 or S_2 occurs first; however, S_4 cannot occur before the occurrence of S_3. It is difficult to wash the body without removing clothes. Based on Definition 3.3.4, the following definitions formulate a *minimal intention* (Definition 3.3.5) as the sufficient condition to reach a goal. Additionally, a minimal intention can be discerned from its gross set (Definition 3.3.6).

Definition 3.3.5. *A minimal intention with respect to a goal g is $I = seq(S_1, S_2, \ldots, S_k)$, where $S_1, \ldots, S_k$ are goal-directed situations for the goal g.*

Definition 3.3.6 (gross set of a minimal intention). *Let I be an arbitrary minimal intention with respect to a goal g. A set of intentions, say $Gross_I$, is said to be the gross set of I if for all $I' \in Gross_I, I'$ and I share the same starting point and ending point to achieve the goal g, and by removing zero or more situations in I', the sequence of I' is exactly the same as that of I. Such situations are called extraneous situations with respect to g. Note that the number of situations in I' is greater than or equal to that of I.*

Definition 3.3.7 (situation dependency). *Let $\sim$ be a binary relation between a set of named situations Sit and intentions. For $S_a \in Sit$ and an arbitrary intention $I_b, S_a \sim I_b$ if and only if the starting point of I_b is S_a.*

Assume that $Gross_I$ denotes a gross set of a minimal intention I and S be a named situation. If $S \sim I$, then for all $I' \in Gross_I, S \sim I'$. This property is employed to define the state space of operational states, say Q. Given a set of minimal intentions Min_Intent and a set of named situations Sit, for any $I \in Min\_Intent$, there exists a situation (the starting point of I) $S \in Sit$ such that $S \sim I$. Let $(S, I') \in Q$, where $I' \in Gross_I$. Then states in Q represent intentions with their initial situations specified. In other words, an operational state shows an intention to be executed. In sum, monitoring human desires helps capture situations in temporal sequences, and the decision to follow a particular intention depends on the corresponding situation at the starting point. The following definition ties all of the aforementioned definitions to form a coherent Situation-theoretic intention-driven service system:

Definition 3.3.8 (transition system of situation-based intentions). *A Transition System of Situation-based Intentions (TSSI) is defined over TempAct, a power set of a set of temporal operators, as a pair (Q, Γ) consisting of the following:*

- *Q, a set of operational states; and*
- *a ternary relation $\Gamma \subseteq (Q \times TempAct \times Q)$, known as the temporal transition of intentions.*

For any operational state $q \in Q$, it is said to be temporally transitioned to q' if there exists an $\alpha \in TempAct$ such that $(q, \alpha, q') \in \Gamma$, written as $q \xrightarrow{\alpha} q'$.

The definitions in this section empower *Situ* to be able to manipulate and compute human intentions through mental states (desires) and context variables. Note that TSSI is a temporal-intention transition system. It is designed to work seamlessly with the conceptual definition of situation and situation dependency relation (Definition 3.3.7) under the Situ framework to infer intention changes. More specifically, they are all built on top of a common temporal mechanism [44].

3.3.2 Software Evolution Triggered by Intention Changes

Based on the intention monitoring methodology, a system is capable of detecting intention changes. For each observation, a system finds intention changes by predicting the transition between desires and examining a sequence of situations; an intention change occurs if the current desire or sufficient condition of an intention is different from the past intentions monitored. Some newly derived desires or intentions are neither predefined nor expected to be found during service runtime. Intention changes are thus categorized into two types: *changes under control* and *changes out of control*. The first type includes the intention changes that are predefined in the system and will typically trigger service optioning, perhaps during runtime. *Changes out of control* are the intention change, which drives the system into service evolution. The second type is detected and handled by the system as follows:

- If the inferred intention dictates a different procedure from past intentions (i.e., procedural incompatibility with any gross set of past/predefined intentions), the system warns system designers of the possibility of an intention change, as well as stores the sequence of observed situations for further analysis through pattern recognition.
- If a desire is newly found in HMM (see Section 4), the system assumes that there is a very new intention because no existing intentions include the new desire. A sequence of observed situations is recorded into a pattern registry in order to register the intention.

After a long period of time, a system performs pattern registration and recognition for the intentions, which are out of control, and formulates those intentions. Service repackaging in the evolution cycle will be triggered when such intentions are detected. The following section suggests a methodology to derive the corresponding requirements specification from the intention specification.

3.3.3 From Intention to Requirements Specification

In this section, we briefly describe a method to develop requirements specification from intention changes (including both old desires with new intentions and new desires). In Human-Computer Interaction studies, a task can be defined as an activity performed to achieve a goal [35]. Similarly, we define task as an atomic temporal operator that triggers transition of situations. In order to satisfy a new desire, a set of *intention specifications* is generated. Intention specification is defined operationally with the following:

1. a signature to identify this intention in the set of operational states defined with TSSI,
2. precondition at the entry of an execution path,
3. the transition of situations along the path, representing a minimal intention (Definition 3.3.5) with the corresponding tasks, and
4. postcondition at the exit of the path.

From each intention specification, a set of tasks, considered as an execution plan, is derived. Such tasks form a task hierarchy [7] to support requirements decomposition and allocation. To satisfy such task specifications, a system invokes the corresponding existing services.

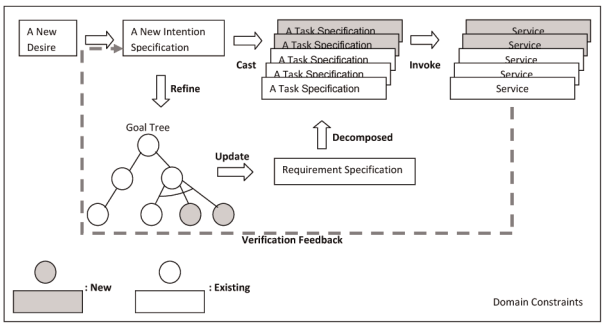

Fig. 5. From intention specification to requirements specification.

For a future version of software, a collection of requirements with respect to those invoked services may not be sufficient for execution plans driven by the intention specification. Fig. 5 depicts a process to update the requirements specification (i.e., a subset of the goal tree which represents a version of service package as discussed in Section 3.2, a derived work flow with task schedules, constraints, data flow, etc.) and repackage services with respect to the subset of the goal tree.

In more precise terms, the refinement process of goal tree is triggered by engaging a new intention specification with the corresponding task specifications, which are customized to achieve a goal targeted. This engagement requires interactions with the system designer or service user in order to affirm a commitment to the new intention specification derived through an inference process in Situ framework. The new intention specification will be cast into a series of task specifications and used to refine the goal tree as deemed necessary. Second, a verification feedback process is to check that the goal adapts to assigned task specifications well by validating execution results of service invocations on the fly. A service package developed around the new intention typically can only focus on achieving their own goals and often lose sight of the overall picture in the system. This can lead to dependency conflicts and erroneous behaviors [45]. Thus, the targeted goal aggregates chosen task specifications and compiles the corresponding service package drawn from a service registry. For example, a customizable UNIX *makefile* can systematically resolve relative dependencies between software packages. Note that each task specification consists of service routines to invoke services, follow schedule, determine priority, and so on. In the context of Web services, a service routine holds parameters to designate Web Service Description Language (WSDL) [46]) ports and bindings in a group of services retrieved from the Universal Description, Discovery and Integration (UDDI) [47]) registry. During the service runtime, the verification feedback from the service package to the new intention specification iteratively compensates semantic gaps between them; if the services invoked at the beginning may not fully satisfy the intention specification, tasks are reconfigured through a user interface of the services. System interactions with system designers and service users will be instrumental to modifying requirements

specification, which accommodates task association rules, schedules, data flow, constraints among tasks, etc.

4 Computational Inference Model—Exploration of Hidden Mental States

The BDI model, which includes three concepts—belief, desire, and intention—was employed to express human mental states. Belief represents the informational level, desire represents the motivational level, and intention represents the execution level of human brains [48]. Intention, as we have shown, can be captured through observing situations. Beliefs and desires can be discovered through questionnaires; however, to build a self-evolving software system, questionnaires are not good solutions since human desires may change over time and sometimes are short-lived.

Generally speaking, human beliefs can be treated as unchanged (or very slowly changed) information because it takes time, even an entire lifetime, to modify beliefs. Assuming that human beliefs are known in advance, we intend to leverage statistical models to predict invisible human mental states (i.e., human desires).

In general, systems developed with human mental states as design concerns can be investigated and analyzed in two different approaches, both of which assume that a set of desire trees, which represents relationships between desires, is derived from beliefs. The first approach (see Section 4.1) simply assumes that a system is initialized with a set of human mental states, a set of visible variables, and a sufficiently large set of historical data of behavioral patterns and environmental contexts. In other words, in developing a system, a set of relevant human mental states is collected. The system designer analyzes the human mental states and historical data to filter out the most relevant visible variables. The second approach (see Section 4.2) assumes that the set of desire trees is the only available knowledge about the system.

Assuming that an explicit duration for observations is predefined when a system is being used, at each point in time, the observation is treated as a vector $\{Variable_1, Variable_2, \ldots, Variable_n\}$, where n represents the number of relevant visible variables considered in the HMM. We define the human mental state at time t as $S(t)$ and the vector $\{Variable_1, Variable_2, \ldots, Variable_n\}$ at time t as $y(t)$, where $t \in N$. The following sections introduce how the *Situ* system works.

4.1 Systems with Historical Data

Generally speaking, historical data help designers gain insight into past human-system behaviors and predict the future trends. To discover the conditional independencies between variables, in probability theory and statistics, Bayesian network [49] is commonly adopted. Nevertheless, Bayesian network cannot model sequences of variables, especially the sequences that are related to time. Human behavior, which usually reflects human mental states, forms certain patterns of actions in time. For our purpose of predicting human mental states, a more dynamic model able to capture changes over time should be considered. With such a model, decision trees can be generalized to dynamic ones, and this property makes it possible to explicitly model

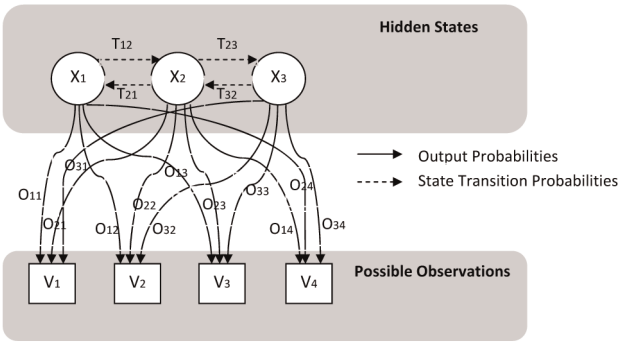

Fig. 6. An illustration for HMM.

sequential decision making and planning under uncertainty. A dynamic Bayesian network [50], which models sequences of variables, can be employed for this purpose. HMM, the simplest dynamic Bayesian network that provides a framework to predict hidden states from observable variables, is considered capable of determining human mental states from human actions [40]. Therefore, HMM is adopted in our work. Two classes of elements, hidden states and observations (an observation consists of a set of visible variables), are in an HMM. Unlike a Markov Model, states in HMM are invisible; however, variables influenced by the states are visible. Due to the invisibility of human mental states, an invisible state in HMM can be readily mapped to a human mental state. Fig. 6 shows the relationship between hidden states and visible variables in HMM.

4.1.1 The Algorithm for HMM

Viterbi Algorithm (VA) is a commonly known and important algorithm for HMM [51]. This section mainly reviews how the VA works and how it is applied to our framework.

The goal of the VA is to find the most probable sequence of hidden states based on the visible observations. Intuitively, with the observations $y(0), y(1), y(2), \ldots, y(K)$, the likelihood of a sequence of states $S(0), S(1), S(2), \ldots, S(K)$ is

$$L = \prod_{t=1}^{K} P(y(t) \wedge (S(t-1) \rightarrow S(t))).$$

That is,

$$L = \prod_{t=1}^{K} P(S(t-1) \rightarrow S(t)) P(y(t) | (S(t-1) \rightarrow S(t)). \quad (1)$$

To derive the most likely sequence of states, the maximum of the likelihood function (1) is computed. For computational purpose, the problem is converted to computing the maximum of $ln(L)$. (Note that $ln(x)$ is a strictly increasing function.) From (1),

$$ln(L) = \sum_{t=1}^{K} m[y(t) : S(t-1), S(t)],$$

where

$$m[y(t) : S(t-1), S(t)] = ln\{P(S(t-1) \rightarrow S(t))\} + ln\{P(y(t) | (S(t-1) \rightarrow S(t)))\}. \quad (2)$$

Let the *State Metric*, $M_K(S_i)$, of state $S_i(K)$ be the maximum over all paths leading from the origin to the ith state at time K:

$$M_K(S_i) = \max_{\text{All paths } S(0)S(1)\ldots S(K-1)} \left\{ \sum_{t=1}^{K-1} m[y(t) : S(t-1), S(t)] + m[y(K) : S(K-1), S_i(K)] \right\}, \tag{3}$$

which can be reformulated as

$$M_K(S_i) = \max_{S_j(K-1)} \{M_{K-1}(S_j) + m[y(K) : S_j(K-1), S_i(K)]\}. \tag{4}$$

Equation (4) is known as the *Viterbi Algorithm*.

Based on (4), it can be derived that for any two points of time, say t_1 and t_2 with $t_1 < t_2$, if we assume that $S(0)$, $S(1), \ldots, S(t_1)$ and $S'(0), S'(1), \ldots, S'(t_2)$ are the Viterbi paths (i.e., the paths of hidden states computed by VA) with respect to the same sequence of visible observations $y(0), y(1), \ldots, y(t_2)$, respectively, the sequence $S'(0)$, $S'(1), \ldots, S'(t_1)$ may not be exactly the same as the sequence $S(0), S(1), \ldots, S(t_1)$. We have modified this feature to fit the proposed framework. For the initialization of a restarted system, the observations should be started first. After a number of observations are collected, VA is applied to search the most likely sequence of human mental states (i.e., human desires). Once the last (current) state is determined, the system derives the next state by computing the most likely transition originating from the current state. The system should allow users to inform the system that his mental state is not the predicted state by interrupting system processes on the fly or inputting certain commands offline. In such cases, the system will restart the prediction process. New hidden states are possible. With the current state being a new state, the next state can be decided by computing the most likely transition originating from the latest existing state.

4.1.2 Detection/Prediction of New Human Mental States

Because the set of human mental states is intrinsically open, HMM with predefined states driving the design of a software system cannot handle the open-ended cases. It is possible that at a point in time, say time t, the actual human mental state does not exist in the predefined set. We have formulated two types of new human mental states.

The first type of new human mental state has unique values on visible variables (i.e., the visible variables can directly imply the new human mental state). For instance, when the visible variables (never observed before) show that the user of a smart home failed to take the correct dose of medication for a period of time, even in the presence of repeated correct reminders, it infers that the user has a new desire (human mental state). This type of new human mental state is relatively easier to be detected compared with another type. It can be determined when the probability for the last state transition

$$\max_{i \in S} P[y(t)|S(t-1) \rightarrow S_i], \text{ range: } [0, 1], \tag{5}$$

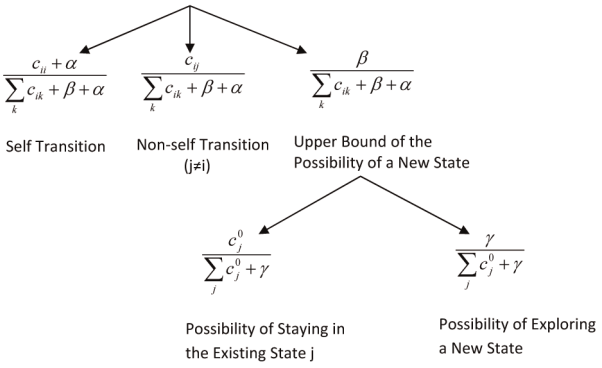

Fig. 7. New state prediction mechanism.

or

$$\max_{i \in S} ln\{P[y(t)|S(t-1) \rightarrow S_i]\}, \text{ range: } (-\infty, 0], \tag{6}$$

where S is the set of existing states, smaller than a threshold.

The second type of new human mental state is one that shares similar values on visible variables with the predefined human mental states. In this case, the system can no longer decide the current human mental state by computing $P[y(t)|S(t-1) \rightarrow S(t)]$ or $ln\{P[y(t)|S(t-1) \rightarrow S(t)]\}$ since those values can be large. To detect this type of new human mental state, a two-level hierarchical Dirichlet Process is applied, shown in a decision tree form in Fig. 7 (see [14]). In Fig. 7, c_{ij} represents the count of transitions from state i to state j in historical state transitions, where $i, j \in \{1, 2, \ldots, |S|\}$, and S is the predefined set of human mental states. c_j^0 represents the count of transitions from any state to state $j, j \in \{1, 2, \ldots, |S|\}$. Note that α presents the tendency of staying in a state, β impacts the possibility of exploring a new state, and γ controls the expected number of hidden states. With these three parameters, various types of possible trajectories exist. The system (or the system designer) should have the ability to learn and tune the parameters to tailor to specific user characteristics.

Therefore, once the system detects a new human mental state, it records the context variables, including all the visible variables, at that moment. For a long period of time, the system designer should continue analyzing the historical data and formulating the new human mental states.

4.2 Systems without Historical Data

This scenario usually occurs when the system is being built or serves as a rapid prototype (or beta version). Normally, it is difficult to train the system without historical data; however, desire trees provide the only known assumptions. We suggest that the system designer should analyze the desire trees and user's daily behaviors so that a Bayesian network with respect to desires can be derived with the actions, context variables, or constraints as conditions. For instance, if the user has the belief that *Health is the best thing in the world*, he will generate the desires related to that. Assume that Fig. 8 is the Bayesian network deduced from a system design by the designer.

Based on Fig. 8, the system can perform top-down (for causal) and bottom-up (from effects to causes) reasoning,

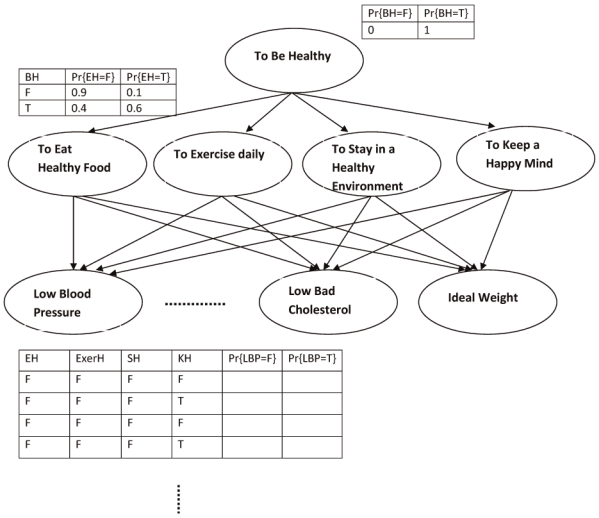

Fig. 8. A Bayesian network with desire tree and influenced contexts.

especially probabilistic inferences, and determine which desire the user has. At each point in time, the system is able to obtain the most likely desire, computed through the values of context variables observed in a timely manner and the Bayesian network provided. Over a long period of time, the system collects enough transitions of desires into the execution history and starts to run with accumulated knowledge about the user and environment.

4.3 Examples of Smart Home

Because of the increasing number of baby boomers turning gray, smart homes/houses are in great demand [52]. The purpose of smart devices used to equip a smart home is to help elderly people who live alone have easier and safer lives. Assume that in the smart home example in which our framework is applied, an elderly woman runs her daily routine. The system was deployed with vendor-supplied training data and keeps accumulating historical data on a daily basis to track her behavioral patterns. At a point in time, say t, she began her daily routine to water the outdoor plants, which does not take more than 10 minutes to complete and return inside. After completing that task, the woman goes to brush her teeth and wash her face. Then, she goes to bed. Her bed can be preheated by either an automatic timer or by manually turning on the switch. This time, she does not come back by the time $t + 20$. Additionally, at the time she went out, she kept the stove on and cooked something. The lights in the kitchen and living room are on. Based on the relevant context variables, the system reacts as follows:

- **Case 1: This kind of event happened before.** In the historical data, visible variables exist whose values are similar to or exactly the same as the context variables (see Fig. 9). Then, the woman's desire (mental state) can be derived by the system. The system detects that her desire makes her stay outside. It turns the stove and lights off for her. Because the regular pattern in the historical data shows that she always brushes her teeth and washes her face before going to bed. The system will ensure that the preheated switch for the bed remains off and waits until she brushes her teeth and washes her face to turn it on and resume the normal routine.
- **Case 2: This kind of event never happened before.** In other words, no record exists in which the relevant context variables are similar or the same as the current values (i.e., $\max_{i \in S} P[y(t)|S(t-1) \rightarrow S_i]$ is close to 0). The system will detect that she has a new desire. Suppose that the system is designed with several lists relevant to the critical events. After the new desire is detected, the system checks the lists with the current context variables and takes the actions based on the lists (i.e., it turns the stove and lights off for her). The system may go as far as to call the nurse station if it detects that the likelihood of hazardous dimension is higher than a predefined threshold. Still, the system will prevent the preheated switch for the bed from engaging and wait until she brushes her teeth and washes her face to resume normal functioning.

Historical Data (Part)

Date	*Visible Variable*	Time 8:30pm	Time 8:40pm	Time 8:50pm	Time 9:00pm	… … …
Jan. 1, 2008	Out	True	True	True	… … …	… … …
	In	False	False	False	… … …	… … …
	Duration (of Staying Outside)	0 (min)	10 (min)	20 (min)	… … …	… … …
	Light (Living Room)	On	On	Pre: On Post: Off	… … …	… … …
	Light (Kitchen)	On	On	Pre: On Post: Off	… … …	… … …
	Stove	On	On	Pre: On Post: Off	… … …	… … …
… … …	… … …	… … …	… … …	… … …	… … …	… … …
Jun. 1, 2008	Out	True	True	True	… … …	… … …
	In	False	False	False	… … …	… … …
	Duration (of Staying Outside)	0 (min)	10 (min)	20 (min)	… … …	… … …
	Light (Living Room)	On	On	Pre: On Post: Off	… … …	… … …
	Light (Kitchen)	On	On	Pre: On Post: Off	… … …	… … …
	Stove	On	On	Pre: On Post: Off	… … …	… … …
… … …	… … …	… … …	… … …	… … …	… … …	… … …
Dec. 30, 2008	Out	True	True	True	… … …	… … …
	In	False	False	False	… … …	… … …
	Duration (of Staying Outside)	0 (min)	10 (min)	20 (min)	… … …	… … …
	Light (Living Room)	On	On	Off	… … …	… … …
	Light (Kitchen)	On	On	Off	… … …	… … …
	Stove	On	On	Pre: On Post: Off	… … …	… … …
	Desire (Hidden State)	To Dump Garbage	To Dump Garbage	To Stay Outside	… … …	… … …

Fig. 9. Part of the historical data for the smart home applications.

This example shows that our framework of adopting HMM facilitates the prediction of human desires. As time goes by, an increasing number of new human mental states (desires) can be identified. The system eventually becomes aged and cannot afford any further adaptation. The software should be evolved to meet the user's needs so that they are able to live in a better environment.

5 Research Challenges

Discovering intention changes on the individual user's basis is a first step toward a major paradigm shift in software development and service evolution. In our mind, *service individualization* is a concept well beyond the prevailing notion of *personalization*. Ultimately, we computer scientists and software engineers want to fulfill our bona fide duty to provide the best, individualized

computer-based services to benefit mankind. Modeling and analyzing human behavioral patterns and mental states are indispensable to the fulfillment of such a duty. In view of this vision, grand research challenges are obviously ahead of us because mankind is intrinsically an open, unpredictable system. We hereby enumerate only a few of the challenges envisioned thus far.

- We assume that when a system is deployed, there will already be a usable repository of predefined sets of desires, situations, and intentions captured from the development organization through questionnaires and experiments. How to collect a minimal set of such information will require many service engineers to engage service requirement elicitation activities and to work with human subjects. Once deployed, the vendor-provided repository of human-centered knowledge must accommodate continuous evolution and growth in order to fulfill the promise for offering "individualized" services. The dynamic and evolving nature of service individualization based on our human-intention-driven framework will demand instant definition and discovering of new services or service variations. Worse, if we resolve to afford an on-demand runtime "service upgrade plan," an end-to-end connectivity to propagate the changes across the entire service system would need to be supported. Even worse, for a community (e.g., a smart housing complex) or a class (e.g., customers with Alzheimer's disease) of service users, newly evolved services may be beneficial to other users in the same group. However, instant propagation of new, improved, or corrected services to all members in a service community will be an acute research challenge. That is, ideally we will need a mechanism to allow customers of the same class be able to discover the emerging service features in a real-time fashion. As such, the service engineers will face tough engineering issues. For example, in the context of Web services, the architecture and management of UDDI service registry [47], despite its current failure to be embraced by mainstream business and industry stakeholders, will need to be significantly enhanced in order to cope with rising expectations. There is also a need for intensive research to help expedite the evolution cycle between an end user, who initiates a new service concept or dictates changes to a service, and a service engineer, who actually makes the changes possible. Success in this research direction may help spark a breakthrough to bring about the final acceptance of UDDI.
- Next, with the assumption that explicit duration for observations is predefined, the system may not completely capture the state transitions of HMM, which may cause the missing of situations. One or more extraneous transitions can always happen between arbitrary two observation points. How to dynamically determine the next observation point is an interesting challenge for researchers. Moreover, in a human-centric service environment, we need to be mindful of the ultrasensitive requirements to avoid inconveniencing the customers when engineering an observation model and deciding the duration, frequency, and rollback/recovery points of observations. For example, we cannot and should not ask the customers of disabilities in an independent-living housing unit to arbitrarily or routinely give permission to repeat an observation in order to enable an embedded HMM model correctly predict hidden mental states.
- The set of visible observation variables plays an important role for the prediction of human mental states. How to select the most critical observation variables is an important research issue, since an inaccurate set of visible variables can result in a biased sequence of human mental states. In addition, the amount of historical data can also impact the derived sequence of human mental states. The investigation of a strategy to maintain an unbiased data set for HMM is inevitable. Certainly researchers should always strive to establish a privacy-preserving observation model.
- As far as system engineers and software developers are concerned, visualization techniques are mostly useful for understanding the largely invisible human mental states in our modeling framework. How do we most effectively present the visual cues to system designers in order to visualize situation changes using visualization tools would be an interesting and challenging research topic.
- Another challenge is the introduction of new human mental states. After a system collects a set of so-called new states (which may include some existing states), the system designer (or system) should be able to formulate new human mental states. A strategy for the formulation of human intentions may require this framework to embrace the emerging *affective computing* [53], [54]. Moreover, the existence of new human mental states influences the prediction of the next state. A methodology to accurately compute the future transition probability from the latest existing state to the next state is required. Since an intention is defined as a temporal sequence of situations to achieve a goal, a strategy to determine the starting point and ending point of a situation is necessary. Without such a strategy, the induction of intentions may derive a wrong intention that may cause adverse problems when systems are running.
- As we have discussed before, after a new intention is identified and a new goal is enacted, an instant requirement has to be determined to achieve the new goal amid the set of desires observed through HMM. An effective method for generating and specifying requirements from the new intention needs to be developed. Service repackaging to connect requirements and services to achieve such a new goal upon discovering a new intention must be performed in an efficient manner (e.g., involving a minimal set of task requirements) and on the fly (i.e., typically while the user is still waiting for the interrupted service to be completed.)
- Finally, in this dynamic service evolution cycle, service level agreement (SLA) becomes a moving target. The services industry will need to investigate the reality of developing the concept of *dynamic*

service level agreement (DSLA) [55]. Structuring services properly at a desirable level of granularity will be key to the success of DSLA research. This represents a step further in automated SLA construction based on semantic Web research [56]. Furthermore, if there can be a DSLA-based service evolution cycle, runtime testing for the evolved software driven by inferred intention changes will challenge many researchers in services computing. Thorough testing of emerging and evolving service packages is essential before deployment; however, completing the evolving, testing, and deployment cycle during service runtime is a very difficult task. Testing related metrics must be studied, and should include adaptability to the evolving human intention, QoS parameters, and situation coverage (i.e., test criteria that cover the key desires of concern, range of context values of relevance, and behavioral observation validity, etc.) In general, dynamic service composition and testing in the context of Web services are difficult research issues. The new service evolution paradigm proposed in this paper will add a new dimension of complexity, where analysis of situations, understanding and monitoring human intentions, and business logic of service rendering must be systematically and carefully matched and cross-checked. However, the added complexity will help the system acquire more powerful features in terms its capability to evolve in runtime.

6 Conclusions

Very few existing software evolution approaches have adequately considered human intention factors in the prevailing research on context-aware and situation-aware research for pervasive computing. A pertinent observation is that software intention [1] is a direct result of a software developer's human intention to serve his users. In other words, software intentions ultimately boil down to human intentions. Human intentions are a root cause in system evolution, which offers a more accurate and timely prediction of future goals for a service system.

This paper presented the *Situ* framework in pervasive computing settings, where context awareness and situation awareness represent the prevailing wisdom of researchers. In our service evolution framework, the well-defined notion of situations helps capture human thinking and behavioral patterns, which, in turn, help developers construct the intention specifications. We further propose that intention specifications help developers complete requirements specification for services and provide concrete means for service definitions and packaging to support services computing. In this process, human intentions drive the changes of goals. In particular, we have discussed two key points as follows:

1. The precise definitions of human intention and situation.
2. A runtime service evolution mechanism for developing successor services.

Due to the pervasive nature of rapid provisioning of new, competitive services in a ubiquitous fashion to leverage the power of computers and Internet, our research focus on human intention changes will prove to be useful and timely. Some observations and speculated research challenges in services computing including Web services have been provided in this paper.

In general, for further improvement of software evolvability, there are two approaches to be considered: short-term evolution (also called "accommodation") and long-term evolution. Short-term evolution is to handle exceptions and to make correct reactions at runtime, while long-term evolution is to monitor a user's behavior and capture new system requirements based on human intentions. Both approaches will be needed to improve the techniques for goal elicitation and inference of human intentions, as in most cases, environmental changes and exceptions caused via user participation must be taken into account. Our *Situ* approach aims to enable short-term evolution with a prediction model of human mental states and facilitate long-term evolution with an "intention-specification to requirements-specification" iterative model.

Acknowledgments

The authors thank the anonymous referees for their insightful comments on an earlier version of this work published by the IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS), which became a draft version for this journal publication. They appreciate the guest editor Elisa Bertino's timely guidance during the preparation of initial submission and final revision. Finally, they thank Laurel Tweed and Hen-I Yang for careful proofreading of the manuscript.

References

[1] R. Laddaga, "Self Adaptive Software—Problems and Projects," *Proc. Int'l Workshop Software Evolvability (SE '06)*, pp. 3-10, 2006.
[2] J. Xia, C. Chang, T. Kim, H. Yang, R. Bose, and S. Helal, "Fault-Resilient Ubiquitous Service Composition," *Proc. Third IET Int'l Conf. Intelligent Environments (IE '07)*, pp. 108-115, 2007.
[3] C. Chang, K. Oyama, H. Jaygarl, and H. Ming, "On Distributed Run-Time Software Evolution Driven by Stakeholders of Smart Home Development," *Proc. Second Int'l Symp. Universal Comm. (ISUC '08)*, pp. 59-66, 2008.
[4] A. Dey, "Understanding and Using Context," *Personal Ubiquitous Computing*, vol. 5, no. 1, pp. 4-7, 2001.
[5] A. Van Lamsweerde and E. Letier, "From Object Orientation to Goal Orientation: A Paradigm Shift for Requirements Engineering," *Proc. Radical Innovations of Software and System Eng., Montery Workshop*, pp. 4-8, 2003.
[6] M.E. Bratman, *Intentions, Plans, and Practical Reason.* Harvard Univ. Press, 1991.
[7] K. Oyama, H. Jaygarl, J. Xia, C. Chang, A. Takeuchi, and H. Fujimoto, "A Human-Machine Dimensional Inference Ontology That Weaves Human Intentions and Requirements of Context Awareness Systems," *Proc. Computer Software and Applications Conf. (COMPSAC '08)*, pp. 287-294, 2008.
[8] H. Ming, K. Oyama, and C. Chang, "Human-Intention Driven Self Adaptive Software Evolvability in Distributed Service Environments," *Proc. 12th IEEE Int'l Workshop Future Trends of Distributed Computing Systems (FTDCS '08)*, pp. 51-57, 2008.
[9] A. Iliasov and A. Romanovsky, "CAMA: Structured Communication Space and Exception Propagation Mechanism for Mobile Agents," *Proc. ECOOP Workshop Exception Handling in Object Oriented Systems: Developing Systems That Handle Exceptions*, 2005.

[10] S. Yau, H. Gong, D. Huang, W. Gao, and L. Zhu, "Specification, Decomposition and Agent Synthesis for Situation-Aware Service-Based Systems," *J. Systems and Software,* vol. 81, no. 10, pp. 1663-1680, 2008.

[11] F. Mastrogiovanni, A. Sgorbissa, and R. Zaccaria, "Understanding Events Relationally and Temporally Related: Context Assessment Strategies for a Smart Home," *Proc. Second Int'l Symp. Universal Comm. (ISUC '08),* pp. 217-224, 2008.

[12] J. Barwise and J. Perry, "The Situation Underground," *Stanford Working Papers in Semantics,* J. Barwise and I. Sag, eds., vol. 1, Stanford Cognitive Science Group, 1980.

[13] V. Morreale, S. Bonura, G. Francaviglia, F. Centineo, M. Cossentino, and S. Gaglio, "Reasoning about Goals in BDI Agents: The PRACTIONIST Framework," *Proc. Workshop Objects and Agents (WOA),* 2006.

[14] M.J. Beal, Z. Ghahramani, and C.E. Rasmussen, "The Infinite Hidden Markov Model," *Advances in Neural Information Processing Systems 14,* 2002.

[15] L.A. Belady and M.M. Lehman, "A Model of Large Program Development," *IBM Systems J.,* vol. 15, no. 1, pp. 225-252, 1976.

[16] M.M. Lehman, J.F. Ramil, P. Wernick, D.E. Perry, and W.M. Turski, "Metrics and Laws of Software Evolution—the Nineties View," *Proc. Fourth Int'l Software Metrics Symp.,* pp. 20-32, 1997.

[17] J. Gorinsek, S. Van Baelen, Y. Berbers, and K. De Vlaminck, "Managing Quality of Service during Evolution Using Component Contracts," *Proc. ETAPS 2003 Workshop Unanticipated Software Evolution (USE '03),* pp. 57-62, 2003.

[18] S. Ciraci and P. van den Broek, "Evolvability as a Quality Attribute of Software Architectures," *Proc. ERCIM Workshop Software Evolution (EVOL '06),* pp. 29-31, 2006.

[19] H. Jaygarl, K. Oyama, J. Xia, and C. Chang, "HESA: A Human-Centric Evolvable Situation-Awareness Model in Smart Homes," *Proc. Int'l Conf. Smart Homes and Health Telematics (ICOST '08),* pp. 153-160, 2008.

[20] T. Strang and C. Linnhoff-Popien, "A Context Modeling Survey," *Proc. UbiComp: The First Int'l Workshop Advanced Context Modeling, Reasoning and Management,* 2004.

[21] P. Bellavista, A. Küpper, and S. Helal, "Location-Based Services: Back to the Future," *IEEE Pervasive Computing,* vol. 7, no. 2, pp. 85-89, Apr.-June 2008.

[22] M. Marschall, "Transforming a Six Month Release Cycle to Continuous Flow," *Proc. Assoc. Geographic Information Laboratories Europe Conf. (AGILE '07),* pp. 395-400, 2007.

[23] H. Levesque, F. Pirri, and R. Reiter, "Foundations for the Situation Calculus," *Electronic Trans. Artificial Intelligence,* pp. 159-178, 1998.

[24] M. Tomasello, M. Carpenter, J. Call, T. Behne, and H. Moll, "Understanding and Sharing Intentions: The Origins of Cultural Cognition," *Behavioral and Brain Sciences,* vol. 28, no. 5, pp. 675-691, 2005.

[25] P.R. Cohen and H.J. Levesque, "Intention Is Choice with Commitment," *Artificial Intelligence,* vol. 42, nos. 2/3, pp. 213-261, 1990.

[26] A.S. Rao and M.P. Georgeff, "Modeling Rational Agents within a BDI-Architecture," *Proc. Second Int'l Conf. Principles of Knowledge Representation and Reasoning (KR),* pp. 473-484, 1991.

[27] R. Bordini, J. Hübner, and R. Vieira, "Jason and the Golden Fleece of Agent-Oriented Programming," *Multi-Agent Programming,* Springer, 2005.

[28] F.R. Meneguzzi and M. Luck, "Composing High-Level Plans for Declarative Programming," *Proc. Workshop Declarative Agent Languages and Technologies,* pp. 69-85, 2008.

[29] M. Móra, J. Lopes, R. Vicari, and H. Coelho, "BDI Models and Systems: Bridging the Gap," *Proc. Workshop Intelligent Agents V, Agent Theories, Architectures, and Languages (ATAL '98),* pp. 11-27, 1999.

[30] R. Scheer, "The Ḿental Stateṫheory of Intentions," *Philosophy,* vol. 79, pp. 121-131, 2004.

[31] R. Kelley, A. Tavakkoli, C. King, M. Nicolescu, M. Nicolescu, and G. Bebis, "Understanding Human Intentions via Hidden Markov Models in Autonomous Mobile Robots," *Proc. Third ACM/IEEE Int'l Conf. Human Robot Interaction (HRI '08),* pp. 367-374, 2008.

[32] A. Cichocki, Y. Washizawa, T. Rutkowski, H. Bakardjian, A. Phan, S. Choi, H. Lee, Q. Zhao, L. Zhang, and Y. Li, "Noninvasive BCIs: Multiway Signal-Processing Array Decompositions," *Computer,* vol. 41, no. 10, pp. 34-42, Oct. 2008.

[33] M. Adcock, J. Chung, and C. Schmandt, "Are We There Yet? User-Centered Temporal Awareness," *Computer,* vol. 42, no. 2, pp. 97-99, Feb. 2009.

[34] D. Norman, *The Design of Everyday Things.* Basic Books, 1988.

[35] Q. Hua, H. Wang, and M. Hemmje, "Conceptual Modeling for Interaction Design," *Proc. 10th Int'l Conf. Human-Computer Interaction,* pp. 351-355, 2003.

[36] C. Rolland, R. Kaabi, and N. Kraïem, "On ISOA: Intentional Services Oriented Architecture," *Proc. Int'l Conf. Advanced Information Systems Eng. (CAISE '07),* pp. 158-172, 2007.

[37] A. Van Lamsweerde, "Goal-Oriented Requirements Engineering: A Guided Tour," *Proc. Fifth IEEE Int'l Symp. Requirements Eng. (RE '01),* pp. 249-262, 2001.

[38] P. Korpipää, M. Koskinen, J. Peltola, S. Mäkelä, and T. Seppänen, "Bayesian Approach to Sensor-Based Context Awareness," *Personal Ubiquitous Computing,* vol. 7, no. 2, pp. 113-124, 2003.

[39] H. Yamahara, F. Harada, H. Takada, and H. Shimakawa, "Dynamic Threshold Determination for Stable Behavior Detection," *World Scientific and Eng. Academy and Soc. (WSEAS) Trans. Computers,* vol. 7, no. 4, pp. 196-206, 2008.

[40] L. Rabiner, "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition," *Proc. IEEE,* vol. 77, no. 2, pp. 257-286, 1989.

[41] O. Saliu and G. Ruhe, "Supporting Software Release Planning Decisions for Evolving Systems," *Proc. 29th IEEE/NASA Software Eng. Workshop (SEW-29),* pp. 14-26, 2005.

[42] Y. Yang and G. Webb, "A Comparative Study of Discretization Methods for Naive-Bayes Classifiers," *Proc. Pacific Rim Knowledge Acquisition Workshop (PKAW '02),* pp. 159-173, 2002.

[43] T. Adam, S. Dadvandipour, and J. Futas, "Influence of Discretization Method on the Digital Control System Performance," *Acta Montanistica Slovaca,* pp. 197-200, 2003.

[44] E.A. Emerson and C.-L. Lei, "Modalities for Model Checking: Branching Time Logic Strikes Back," *Science of Computer Programming,* vol. 8, pp. 275-306, 1987.

[45] H. Yang, J. King, S. Helal, and E. Jansen, "A Context-Driven Programming Model for Pervasive Spaces," *Proc. Int'l Conf. Smart Homes and Health Telematics (ICOST '07),* pp. 31-43, 2007.

[46] *WSDL 2.0 Specification,* http://www.w3.org/TR/wsdl20-primer, 2009.

[47] *UDDI Version 3 Specification,* http://uddi.org/pubs/uddi_v3.htm, 2009.

[48] G. Weiss, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence.* MIT Press, 2000.

[49] A. Mittal and A. Kassim, *Bayesian Network Technologies: Applications and Graphical Models.* IGI Global, 2007.

[50] K. Korb and A. Nicholson, *Bayesian Artificial Intelligence.* CRC Press, 2004.

[51] D. Hernando, V. Crespi, and G. Cybenko, "Efficient Computation of the Hidden Markov Model Entropy for a Given Observation Sequence," *IEEE Trans. Information Theory,* vol. 51, no. 7, pp. 2681-2685, July 2005.

[52] R. Babbitt, J. Wong, C. Chang, and S. Mitra, "Privacy Management in Smart Homes: Design and Analysis," *Proc. Int'l Conf. Aging, Disability, Independence (ICADI '06),* pp. 55-64, 2006.

[53] R. Picard, *Affective Computing.* MIT Press, 1997.

[54] X. Li, "Integrating User Affective State Assessment in Enhancing Hci: Review and Proposition," *The Open Cybernetics and Systemics J.,* pp. 192-205, 2008.

[55] J. Ejarque, M. de Palol, I. Goiri, F. Julia, J. Guitart, R. Badia, and J. Torres, "SLA-Driven Semantically-Enhanced Dynamic Resource Allocator for Virtualized Service Providers," *Proc. IEEE Fourth Int'l Conf. eScience,* pp. 8-15, 2008.

[56] A. Paschke, "RBSLA—A Declarative Rule-Based Service Level Agreement Language Based on RuleML," *Proc. Int'l Conf. Intelligent Agents, Web Technology and Internet Commerce (IAWTIC '05),* pp. 308-314, 2005.

Carl K. Chang received the PhD degree in computer science from Northwestern University in 1982. He is currently a professor and chair of the Department of Computer Science at Iowa State University. He worked for GTE Automatic Electric and Bell Laboratories before joining the University of Illinois at Chicago in 1984. He joined Iowa State University in 2002. His research interests include requirements engineering, software architecture, net-centric computing, and services computing. He was the president of the IEEE Computer Society in 2004. Previously, he served as the editor-in-chief of *IEEE Software* from 1991 to 1994. He received the IEEE Computer Society's Meritorious Service Award, Outstanding Contribution Award, the Golden Core recognition, and the IEEE Third Millennium Medal. In 2006, he received the prestigious Marin Drinov Medal from the Bulgarian Academy of Sciences and was recognized by IBM with the IBM Faculty Award in 2006, 2007, and 2009. In January 2007, he became the editor-in-chief of *Computer*, the flagship publication of the IEEE Computer Society. He is a fellow of the IEEE and the AAAS.

Hsin-yi Jiang received the bachelor's degree in mathematics from the National Taiwan Normal University and the master's degree in applied mathematics from the National Chung Cheng University in Taiwan. She is currently working toward the PhD degree in the Department of Computer Science at Iowa State University. Her research interests include software engineering, evolutionary computation, and pervasive computing.

Hua Ming received the bachelor of science degree in computer science from the University of Science and Technology Beijing. He joined the Department of Computer Science at Iowa State University in 2003 and received the master of science degree in computer science in 2007 before working toward the PhD degree. His research interests include the interface among the areas of software engineering, service computing, distributed systems, and intelligent agents.

Katsunori Oyama received the BS, ME, and PhD degrees in computer science from Nihon University, Japan, in 2001, 2003, and 2007, respectively. He is currently a postdoctoral researcher in the Department of Computer Science at Iowa State University. He received the Special Model Award in 2005 and the Silver Model Award in 2007 for having attended the Embedded Technology Software Design Robot Contest (http://www.etrobo.jp). His primary research interests concentrate on design thought process, ontology engineering, object-oriented analysis and design, situation-awareness, and software evolution. He is a member of the IPSJ.