Chapter 40
Incorporating Users into AmI System Design: From Requirements Toward Automation

Estefanía Serral, Luca Sabatucci, Chiara Leonardi, Pedro Valderas, Angelo Susi, Massimo Zancanaro, and Vicente Pelechano

1 Introduction

The term ambient intelligence (AmI) is still a vision of the future of consumer electronics in which the computational power is embedded in everyday appliances and physical objects to turn environments into sensitive places able to understand users' needs and to automate their daily tasks (Weiser 1995). In the context of AmI, task automation is central and raises many challenges since the system must adapt to each individual's specific needs. These challenges become even more critical when the domain is characterized by the presence of many actors, every one owning different institutional roles, responsibilities, skills, and motivations (Cook et al. 2003). In addition, since users' preferences may change in time, it is also important that the developed system provides evolution facilities for adapting to new requirements; otherwise, the system may become useless, obsolete, or perceived as intrusive by final users. It is therefore of paramount importance to use requirements engineering techniques for the analysis of users' needs and for involving users to participate in design and development choices (Rolland and Salinesi 2009, Van Lamsweerde 2003).

We present the results of a cooperative work for developing a user-intensive AmI system that involves a multidisciplinary team composed by, among others, software engineers, sociologists, and HCI designers. The challenges are summarized in (1) eliciting a set of requirements that encapsulate the real users'

E. Serral (✉) • P. Valderas • V. Pelechano
Centro de investigación en Métodos de Producción de Software,
Universidad Politécnica de Valencia, Valencia, Spain
e-mail: eserral@pros.upv.es; pvalderas@pros.upv.es; pele@pros.upv.es

L. Sabatucci • C. Leonardi • A. Susi • M. Zancanaro
Fondazione Bruno Kessler IRST, Trento, Italy
e-mail: sabatucci@fbk.eu; cleonardi@fbk.eu; susi@fbk.eu; zancana@fbk.eu

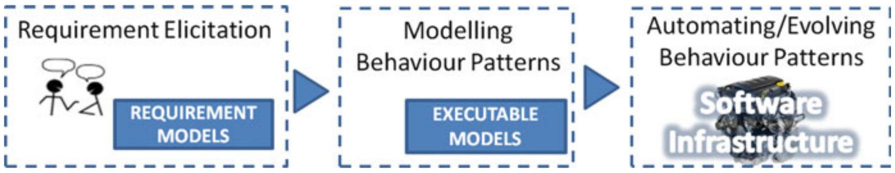


Fig. 40.1 Overview of the methodology

needs and (2) providing software infrastructures that guarantee the fulfillment of requirements and enable their customization and evolution at runtime.

The receipt we present in this chapter contains three main ingredients: a goal-oriented requirement engineering (GORE) process (Bresciani et al. 2004), a user-centered design process (Leonardi et al. 2010a) and a software infrastructure based on executable models (Serral et al. 2010a). Opportunely mixing these ingredients, we obtain:

- Iterative involvement of users within the design methodology, from early requirements identification to prototype evaluation
- Requirements engineering techniques able to integrate users' needs, preferences, and activities since the beginning of the design process
- Software infrastructures that guarantee the fulfillment of these requirements and also simplify their evolution over time without the need to redeploy the system

The resulting methodology (Fig. 40.1) consists of the following steps:

Requirement elicitation: This step stems from the consideration that modelling user requirements requires a deep understanding of the organizational and social context in which any system will operate (Nuseibeh and Easterbrook 2000). Section 40.3 provides details about the RE process, which is obtained by coupling two methods (Leonardi et al. 2010a,b): the User-Centered Design (UCD) (Cooper et al. 2007, Sharp et al. 2007) and Tropos (Bresciani et al. 2004) (a GORE process).

Identifying and modeling behavior patterns: A behavior pattern is a set of tasks that are habitually performed when similar contexts arise (Neal and Wood 2007). In this step, the behavior patterns that users want to be automated are identified and specified in models of a high level of abstraction, which are also prepared for being directly executed. Section 40.4 introduces the language to specify these executable models. Section 40.5 explains the process to obtain the executable models (behavior patterns) from the requirement elicitation models.

Automating and evolving user behavior patterns: We have developed a software infrastructure that directly uses the executable models at runtime to automate the behavior patterns as specified. Section 40.6 introduces the infrastructure and discusses automation and evolution of behavior patterns.

The feasibility of this methodology and the automation infrastructure has been evaluated through the development of an automated user-intensive AmI system within the ACube Project.[1] It aims at providing support to medical and assistance staff in elderly. This is also the running example used overall this chapter.

Finally, Sect. 40.7 concludes this chapter providing discussions about the benefits and drawbacks of the presented work.

2 Related Work

AmI is a young research area that has been mainly focused on the development of implementation technologies, but, recently, methodological approaches are emerging. So far, the integration of requirement elicitation techniques with software infrastructures that guarantee the fulfillment of the captured requirements is still a challenge. Here, we analyze the state of art about RE and automation approaches for AmI systems.

Kolos-Mazuryk et al. present a survey of GORE techniques used for AmI system development. In this study, they identify the modeling of context and interactions is still an open challenge. Our research is actually moving in this direction. Sutcliffe et al. (1998) propose a framework for relating scenarios to use cases. Also, works such as Dardenne et al. [1993], Rolland and Salinesi [2009], Uchitel et al. [2004], and Cockburn [2001]) propose combinations of goal techniques with scenarios. Casas et al. (2008) describe how they use *personas* in a project focused on the design of AmI systems. Their work suggests the usefulness of personas as a starting point for user modeling and discusses implications for using this information in the design of the architecture of the system. In addition, our approach tries a step further for providing an integrated effort, covering as well as requirement elicitation, all the other stages of the development.

Concerning the automation topic, two interesting machine-learning approaches are used in the MavHome (Cook et al. 2003) and iDorm (Hagras et al. 2004) projects. These are respectively based on probabilistic and fuzzy approaches, techniques that suffer of the typical training problems that are often not practicable in real contexts. Also, these approaches act on the basis of what happened, according to what they see happening and believe is going to happen, without considering users' desires or knowing users' goals.

3 Requirement Elicitation

The requirement elicitation process is based on the systematic collaboration of Tropos and UCD, with the aim of taking advantages of the synergy of the two approaches. UCD methods (specifically contextual inquiries, scenarios, and

[1] The ACube project was founded by the local government of the Autonomous Province of Trento in Italy;http://acube.fbk.eu.

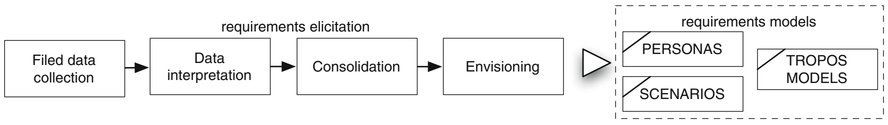


Fig. 40.2 Requirement elicitation process

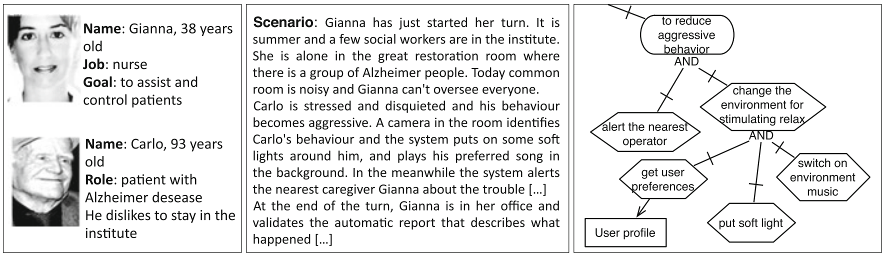
personas) (Cooper et al. 2007, Dey 2001, Rolland and Salinesi 2009) are used to gain a rich understanding of the domain and to efficiently communicate and negotiate design ideas with stakeholders. On the other side, Tropos models are used to gain an abstract model of the observed domain, and to reason on strategical details for introducing the system.

The requirement elicitation (as shown in Fig. 40.2) is composed of the following four phases, in which techniques from Tropos and UCD are combined:

Field Data Collection – Contextual Inquiry. The initial field data collection activity is performed for acquiring the richest possible understanding of the domain. The first step for a successful design of a product is actually to consider the wide range of stakeholders (Sharp et al. 2007). UCD approach recommends to early define which are the users of the system and their characteristics and to define how to involve them in the design process. Several UCD methods exist in order to get rich insights about the domain. Recently, contextual inquiry (Sharp et al. 2007) demonstrated the capacity to satisfy the needs for a deep, but at the same time rapid, understanding of complex domain. Contextual inquiry mainly consists in interviewing people in their context, preferably when performing their tasks. Resulting data are raw, since it is preferable at this stage to keep the richness of the data and avoid abstraction.

Data Interpretation. The *Tropos Early Requirement* is used for modeling the initial set of domain entities when the system is not yet existing. It includes a bird's-eye view over the domain, in which actors and roles are specified together with their responsibilities and delegations. This view provides an intuition of which interactions occur in the environment. Subsequently each actor is exploited in a goal model, in order to provide details about human behavior, highlighting the rationale by relating each activity to institutional motivations. An example of strategic-view data is the delegation of responsibility [to ensure security in the institute] that is assigned to caregivers. This institutional goal is declined in two caregiver goals: [to avoid guest's aggressive behavior] and [to avoid guest's escape]. Details of these two goals are provided in caregiver's goal model, in which it is defined that the plans to achieve both of them are [direct observation of critical guests] and [observation of critical spaces].

Data Consolidation. The consolidation activity concludes the analysis of the domain by generating two artifacts: personas and scenarios. Their conjunct use increases the ability to identify problems and exceptional cases (Sutcliffe et al. 1998) and to envisage the system (Rolland and Salinesi 2009). *Personas* are descriptive models of system-to-be users based on behavioral data, derived from patterns observed during



Name: Gianna, 38 years old
Job: nurse
Goal: to assist and control patients

Name: Carlo, 93 years old
Role: patient with Alzheimer desease
He dislikes to stay in the institute

Scenario: Gianna has just started her turn. It is summer and a few social workers are in the institute. She is alone in the great restoration room where there is a group of Alzheimer people. Today common room is noisy and Gianna can't oversee everyone.
Carlo is stressed and disquieted and his behaviour becomes aggressive. A camera in the room identifies Carlo's behaviour and the system puts on some soft lights around him, and plays his preferred song in the background. In the meanwhile the system alerts the nearest caregiver Gianna about the trouble […]
At the end of the turn, Gianna is in her office and validates the automatic report that describes what happened […]

to reduce aggressive behavior
AND
alert the nearest operator
change the environment for stimulating relax
AND
get user preferences
switch on environment music
User profile
put soft light

Fig. 40.3 Three produced artifacts: a couple of relevant personas, the scenario of *aggressive behavior* in which they are involved, and the slice of correspondent goal model

interviews, with the aim of representing the diversity of observed motivations, behaviors, and mental models (Cooper et al. 2007). Examples can be found in Fig. 40.3. Personas are provided in the form of rich descriptions of archetype users, meant to focus attention on users goals and motivations both for envisioning and validation purposes. Personas are meant to trigger on emphatic response from the designers and users, in order to support them in taking design decisions at both the cognitive and emotional level.

Envisioning. The envisioning phase aims at specifying the system-to-be and its impact to the domain. Many techniques can be profitable for envisioning system functionalities and services; examples are internal meetings, brainstorming, focus groups, and scenarios. The outcomes of this phase are the list of requirements (modeled in the Tropos language) and a set of technological scenarios.

Tropos Late Requirements. Whereas the early requirement focuses on the domain as it is, this phase introduces the system as a new actor of the domain. In particular, some responsibilities (currently assigned to users) will be delegated to the *system*, while other new dependencies are discovered among actors (both human and software) of the domain. Specific emphasis is given to modeling why the system is necessary and how it will modify current human practices. This enables the analysis of the system impact to the social organization and to the actors. Finally, the system actor is deeply analyzed in order to specify how it will accomplish the goals that are delegated by the humans. For instance, the [direct observation of critical guests] (identified in the early requirements phase) is delegated to the system, and it is decomposed into [to reduce dangerous situations] and [to reduce aggressive behavior]. In particular, designers defined that the system plans for achieving the goal [to reduce aggressive behavior] are (see Fig. 40.3) the following: [alert the nearest caregiver] and [change environment parameters] (e.g., put soft light and play music).

Technological Scenarios. Scenarios are short narrative stories that represent personas in their context, supported by the envisaged technology (see Fig. 40.3). Scenarios concretely describe the behavior of services as experienced by specific, though fictional, users. Stories help the design teams in negotiating a shared representation of the domain and hence a more effective collaborative elicitation of requirements.

4 Identifying and Modeling Users' Behavior Patterns

The behavior patterns to be automated must be identified and modeled from the obtained requirements. To do this, the requirement models are transformed into two executable models: a context model (which specifies the context on which the behavior patterns to be automated depend), and a task model (which describes the tasks that must be carried out for each behavior pattern according to the context described in the context model). Before discussing the transformation, we briefly describe these target models. More details about them can be found in Serral et al. [2010a].

ContextModel. The context model semantically describes the context for properly automating behavior patterns. This model is based on a context ontology proposed in Serral et al. [2010b]. The ontology provides classes such as *User*, which captures information about the users; *Policy*, which captures the system permission for each user; *Location*, which captures information about the locations of the environment where the system is deployed; *EnviromentProperty*, which captures the values of the properties sensed in the environment; and *TemporalProperty*, which captures temporal information. Some of the most important classes of this ontology are also shown on the left of Fig. 40.4. The specific context of the system is represented as instances of the ontology classes. For instance, the *restoration room* should be defined as an instance of the *Location* class, the *noise level* should be defined as an instance of the *EnviromentProperty* class, etc. On the right of Fig. 40.4, an example of context model in a tree form is shown.

Context-Adaptive Task Model. The context-adaptive task model allows the behaviour patterns to be specified by splitting them up into simpler tasks whose execution adapt to the context. We have selected a task model mainly for two reasons: (1) tasks center the requirements modeling process around the user's own experiences (Lauesen 2003), and (2) it allows great expressivity (Johnson 1999), which is needed for accurately specifying user behavior patterns in such a way that they can be automated from their specification.

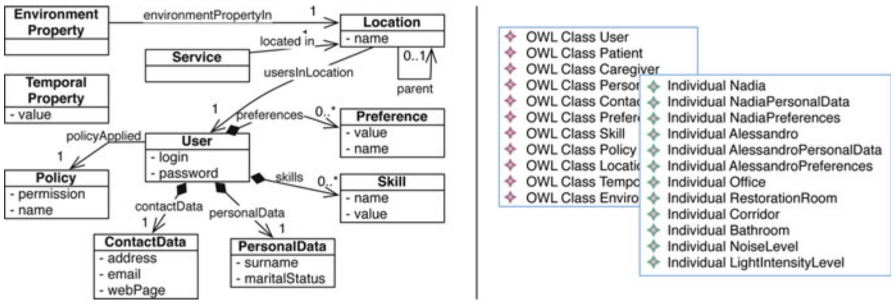


Fig. 40.4 Example of context model

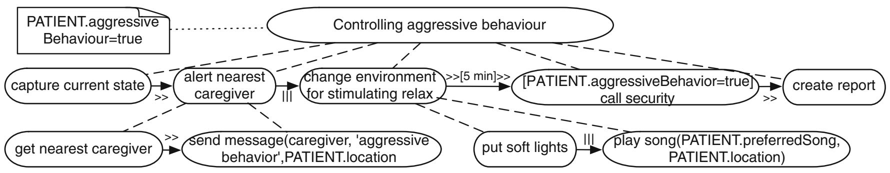

PATIENT.aggressive Behaviour=true
Controlling aggressive behaviour
capture current state
alert nearest caregiver
change environment for stimulating relax
>>[5 min]>>
[PATIENT.aggressiveBehavior=true] call security
create report
get nearest caregiver
send message(caregiver, 'aggressive behavior',PATIENT.location
put soft lights
play song(PATIENT.preferredSong, PATIENT.location)
>>
|||

Fig. 40.5 Examples of behavior pattern modelling

The proposed context-adaptive task model is based on the *hierarchical task analysis (HTA)* technique (Shepherd 2001), which breaks down tasks hierarchically into other tasks. We propose to define a task hierarchy for each behavior pattern. Figure 40.5 shows, for instance, the *controlling aggressive behavior* pattern. The root task represents the behavior pattern and has an associated context situation, which defines the context conditions whose fulfilment enables the execution of the behavior pattern. It also has a priority (high, medium, and low) to establish the execution priority of the pattern in case several patterns are enabled at the same time. This root task can be broken down into composite tasks (which are intermediate tasks) and/or system tasks (which are leaf tasks). The composite tasks are used for grouping subtasks that share a common behavior or goal. The system tasks represent atomic tasks that have to be performed by the system (e.g., play a song). Hence, each system task has to be related to a pervasive service that can carry it out. The relation is established by means of the name of the service and the name of its corresponding operation (e.g., multimedia service and playMusic operation). Also, system tasks can have output and input parameters, which can be context parameters.

Both composite and system tasks can have a context precondition that are represented between square brackets. If the precondition is not fulfilled, the task will not be executed. In addition, every task has a name (which explains the task in a user-comprehensible way) and an internal ID (which is a unique identifier).

A behavior pattern or a composite task is broken down into simpler tasks that are related between them using temporal operators that determine the task execution order. This is known in HTA as a plan (Shepherd 2001). We base the temporal operator definition on (Paternó 2002) which provides one of the richest sets of temporal operators. For instance, the $|=|$ relationship means that the related tasks can be performed in any order, the $\gg$ relationship means that the target task will be executed when the first finishes, and the $t \gg$ relationship means that the target task is enabled after t minutes. These temporal relationships can be also used between behavior patterns to allow the composition of patterns.

Note that we make the specified tasks automatically adapt to context by using context parameters. To refer a context parameter, the name of the context property and the name of the individual to which this property belongs have to be specified (e.g., the *aggressive behavior* context property of the *Carlo* individual). However, to deal with the automation of behavior patterns for a group of users, instead of specifying the context property of a user individual, it can be specified as a context property of the ontology class that groups together the corresponding user individuals. For instance, the *controlling aggressive behavior* pattern has to be executed for every patient in which aggressive behavior is detected; therefore, instead of specifying the same behavior pattern for each patient, we specify the behavior pattern once and use in its context situation the *aggressive behavior* context property of the *PATIENT* class, indicating by using capital letters that it is an ontology class, and therefore, the context condition has to be checked for every *patient* individual.

5 From Requirement Models to Executable Models

The methodology we propose is completed by a set of guidelines which allow context and task models to be created from personas, scenarios, and Tropos models. The activity cannot be automated because it cannot leave aside interpretation and reasoning. It is also possible that moving from requirements to executable models, the designer discovers lacks and incongruences; this is solved by starting a new contextual inquiry iteration, by going back to final users with new questions.

Step 1: Detect the behavior patterns to be automated. The step consists in identifying the behavior patterns that can be automated by the system. To identify them, the Tropos goal model is used. A one-to-one relationship is identified between goals delegated to the system and behavior patterns (e.g., the goal [to reduce aggressive behavior] could be transformed into a behavior pattern named *controlling aggressive behavior*).

Step 2: Model the task hierarchy of each behavior pattern. Each behavior pattern is specified using a task hierarchy, from more general to more specific tasks. This hierarchy is obtained from the task decomposition of the corresponding goal in the Tropos goal model. This is completed with the information provided by the technological scenarios: the action verbs whose subject is the system represent tasks to be automated (e.g., *the system plays his (Carlo) preferred song*). An example of task hierarchy obtained following this guideline is shown in Fig. 40.6.

Step 3: Specify users. The users involved in the tasks to be automated are identified. The Tropos actor model and personas provide useful information for creating a hierarchy of users, which has to be specified in the ontology as subclasses of the *User* class. For instance, the actor model identifies the roles *caregiver* and *patient*, while the personas instrument identifies more specific type of users: Carlo, who is a *patient with Alzheimer disease*, and Gianna, who is a *nurse* which is a type of caregiver. Real users will be specified in the hierarchy as individuals of the class that better represent their characteristics.

Step 4: Specify context. Tasks to be automated usually depend on context information. This context information appear in the scenarios as adjectives (e.g., *noisy*), locations (e.g., *restoration room*), temporal aspects (e.g., *season*), etc. Also, the motivations of the goals specified in the Tropos goal model can be used for detected

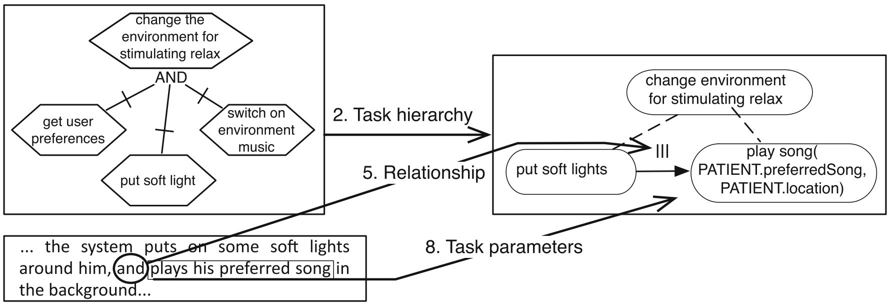

change the environment for stimulating relax
AND
get user preferences
switch on environment music
put soft light
2. Task hierarchy
5. Relationship
change environment for stimulating relax
put soft lights
|||
play song(PATIENT.preferredSong, PATIENT.location)
8. Task parameters
… the system puts on some soft lights around him, and plays his preferred song in the background…

Fig. 40.6 Transformation performed following the provided guidelines

needed context information (e.g., *aggressive behavior*). The identified context properties must be specified in the context model as individuals of the corresponding ontology classes (e.g., noisy should be an instance of the EnvironmentProperty class).

Step 5: Specify temporal relationships. If a behavior pattern, or a composite task, has been refined by temporal refinements, its subtasks have to be related between them by using temporal relationships that rigorously specify the execution order of these subtasks. Scenarios can help to define these relationships. For instance, as shown in Fig. 40.6, in the scenario it is explained that the systems put soft lights and play Carlo's preferred song; meanwhile, the system alerts the caregivers. From this information, we can deduce that the order of execution of these tasks is not important; then, the $|=|$ relationship must be used.

Step 6: Specify the context situation. Each behavior pattern has to be related with a context situation whose fulfillment activates the execution of the pattern. The meaning of the goal to be achieved as well as the technical scenarios can help to define these context situations. For instance, to achieve the goal [to reduce aggressive behavior] , the identified controlling aggressive behavior pattern must be activated when an aggressive behavior is detected, as also is explained in the scenario shown in Fig. 40.3 (*…his behavior becomes aggressive. A camera in the room identifies it and the system …*).

Step 7: Specify context dependencies. The specified tasks may have to be executed only when some context conditions are satisfied. Thus, these conditions are specified as task preconditions by using the context properties identified in Step 3. For instance, the *call security* task will be executed if the user continues behaving aggressively after executing the previous tasks; then, the context precondition $aggressiveBehavior = true$ must be added to this task.

Step 8: Specify task parameters. If a system task needs parameters to be performed. To detect these parameters, resources in goal models and technological scenarios are used. An example from the scenarios is shown in Fig. 40.6: the text *the system plays his (Carlo) preferred song*, is used for detecting the 'PATIENT. preferredSong' parameter of the task *play song*.

6 Automating and Evolving Behavior Patterns

In this section, we introduce a software infrastructure that allows the specified user behavior patterns to be automated. The infrastructure interprets the task model and the context model at runtime and executes the behavior patterns in the opportune context. This infrastructure (shown in Fig. 40.7 and presented in detail in Serral et al. 2010a) is defined by the following elements, which are implemented by using Java and OSGi technology:

Pervasive Services. A smart environment provides users with pervasive services that control the devices deployed in the environment (e.g., switching lights on, playing music, etc.) and sense context (e.g., detection of presence, measurement of temperature, etc.). We have used a model-driven development (MDD) method (Serral et al. 2010b) to automatically generate the Java/OSGi code of these services from a set of models. The implementation of these services is out of the scope of this work.

Model Management Mechanisms. Provide high-level constructors for managing the models at runtime. Using these mechanisms, any element of the task model and any individuals of the context model can be managed. These mechanisms can be downloaded from http://www.pros.upv.es/art/.

Context Monitor. Context changes are physically detected by sensors, which are controlled by the pervasive services. The context monitor is continuously monitoring these services to capture the context changes. When a change is detected, the monitor processes it and updates the context model accordingly using the model management mechanisms. For instance, if aggressive behavior in a user is detected, the *aggressivenbehavior* context property of the corresponding *User* individual is updated by the context monitor, which also informs the automation engine (next explained) about this update.

Automation Engine. Is in charge of automating the specified behavior patterns in the opportune context. When the engine is informed about a context update, it analyzes the task model to check if some behavior pattern must be executed in the new context. If so, the engine performs the corresponding behavior patterns respecting their priorities. To perform each behavior pattern, the engine executes its system tasks according to

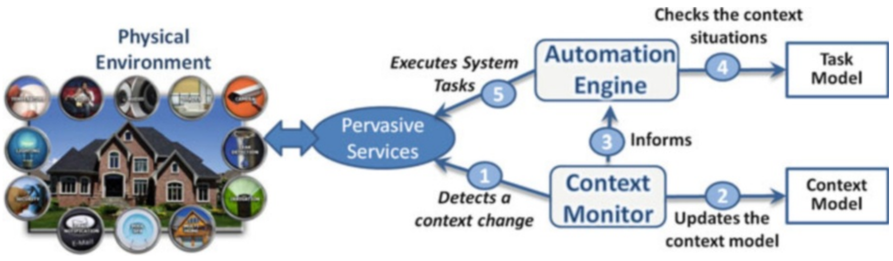


Fig. 40.7 Software infrastructure for automating user behaviour patterns

the temporal relationships specified among its tasks and the current context information (stored in the context model) on which tasks and relationships depend. To execute each system task, the engine searches for the service associated to that task and executes it with the corresponding parameters. For instance, if the context situation of the *controlling aggressive behavior* pattern is fulfilled (see Fig. 40.5), the system captures the current context state. Then, the system searches for the caregivers nearest to the patient location and then sends them a message to warn that aggressive behavior has been detected in the corresponding location. Next, the system puts soft lights and plays the preferred song of the patient that is behaving aggressively. Five minutes later, if the patient is still behaving aggressively, the system warns the security officers. Finally, the system creates a report about the incident.

It is worth noting that rather than translating the models into code, the automation engine directly interprets them at runtime to automate the behavior patterns as specified. This considerably facilitates the evolution of the patterns at runtime: as soon as the models are changed, the changes are applied in the system. To carry out this evolution, the model management mechanisms can be easily used since they provide concepts of a high level of abstraction (e.g., preference, task, behavior pattern, etc.).

7 Discussion

In this work, we have presented a novel methodology for developing user-intensive AmI systems capable of automating the behavior patterns of their users. This methodology supports all the stages of the development process. To achieve this, the methodology integrates a requirement elicitation process that combines UCD and GORE techniques with a software infrastructure that automates the behavior patterns identified in the automation requirements. The feasibility of the approach has been successfully evaluated by developing the ACube case study. In this case study, four scenarios, five personas, and a Tropos model were specified, thus producing 73 requirements among functional and nonfunctional ones. From these requirements, four behavior patterns were obtained and specified using the context and task models. Using these models, the behavior patterns were correctly automated by the provided software infrastructure. Detailed information about this evaluation can be found in [report].

Benefits and Drawbacks. The proposed methodology has been conceived for AmI settings in which the presence of humans is relevant and the difficulty is to build a believable knowledge of the domain and to precisely identify users' needs. The proposed approach is grounded over social science techniques that reduce the gap between analysts and the observed domain: the knowledge is directly extracted with users' participation to discover their real needs. In particular, the methodology is suitable for a quick prototyping approach, in which the development is supported by frequent deployment of mock-ups and prototypes to submit to users' validation.

An additional strength of the approach is the high level of customization of the infrastructure that allows for design-time adapting the system to different setting, for instance, nursing homes with different services. In addition, the model interpretation strategy considerably facilitates the runtime evolution of the automated behaviour to adapt it to changes in users' needs. Mechanisms and tools inspired by end-user techniques have been developed to allow this evolution (?); however, it has to be still performed by users. To achieve a more automatic evolution, we plan to extend the provided infrastructure with machine-learning algorithms that automatically detect the adaptations that must be performed in the system and change the models accordingly if users so desire.

Finally, although the approach has been evaluated by applying it into a real case study, more experimentation is needed. Thus, we also plan to apply the approach in several case studies of different domains.

References

Bresciani P, Perini A, Giorgini P, Giunchiglia F, Mylopoulos J (2004) Tropos: an agent-oriented software development methodology. In: Proceedngs of the AAMAS. IEEE Computer Society, Los Alamitos, pp 203–236

Casas R, Blasco Marín R, Robinet A, Delgado A, Yarza A, McGinn J, Picking R, Grout V (2008) User modelling in ambient intelligence for elderly and disabled people. In: Proceedngs of the computers helping people with special needs. Springer, Berlin/New York, pp 114–122

Cockburn A (2001) Writing effective use cases, vol 1. Addison-Wesley, Boston

Cook DJ, Youngblood M, Heierman IEO, Gopalratnam K, Rao S, Litvin A, Khawaja F (2003) Mavhome: an agent-based smart home. In: Proceedings of the PerCom. IEEE Computer Society, Los Alamitos, pp 521–524

Cooper A, Reimann R, Cronin D (2007) About face 3: the essentials of interaction design. Wiley, Indianapolis

Dardenne A, Lamsweerde A, Fickas S (1993) Goal-directed requirements acquisition. Sci Comput Program 20(1–2):3–50

Dey AK (2001) Understanding and using context. PUC

Hagras H, Callaghan V, Colley M, Clarke G, Pounds-Cornish A, Duman H (2004) Creating an ambient-intelligence environment using embedded agents. IEEE Intel Syst 19(6):12–20

Johnson P (1999) Tasks and situations: considerations for models and design principles in human computer interaction. In: Proceedings of the HCI international. Lawrence Erlbaum, Mahwah/London, pp 1199–1204

Kolos-Mazuryk L, Eck P, Wieringa R A survey of requirements engineering methods for pervasive services. In: Proceedings of the workshop on building software for pervasive computing, OOPSLA'05

Lauesen S (2003) Task description as functional requirements. IEEE Softw 20:58–65

Leonardi C, Sabatucci L, Susi A, Zancanaro M (2010a) Ahab's leg: mediating semi-formal requirement to final users. In: Proceedings of the CAiSE'10, Hammamet

Leonardi C, Sabatucci L, Susi A, Zancanaro M (2010b) Exploring the boundaries: when method fragmentation is not convenient. In: Proceedings of the IEEE FIPA workshop on design process documentation and fragmentation, Lyon

Neal DT, Wood W (2007) Automaticity in situ: the nature of habit in daily life. In: Psychology of action: mechanisms of human action, vol 2

Nuseibeh B, Easterbrook S (2000) Requirements engineering: a roadmap. In: Proceedings of the conference on the future of software engineering. ACM, New York, pp 35–46

Paternó F (2002) ConcurTaskTrees: an engineered approach to model-based design of interactive systems. Lawrence Erlbaum Associates

Rolland C, Salinesi C (2009) Supporting Requirements Elicitation through goal/scenario coupling. In: Conceptual modeling: foundations and applications. Springer, Berlin, p 416

Serral E, Valderas P, Pelechano V (2010a) Improving the cold-start problem in user task automation by using models at runtime. In: Proceedings of the ISD'10. Springer, pp 648–659

Serral E, Valderas P, Pelechano V (2010b) Towards the model driven development of context-aware pervasive systems. PMC 6(2):254–280

Sharp H, Rogers Y, Preece J (2007) Interaction design: beyond human computer interaction. Wiley, Chichester/Hoboken

Shepherd A (2001) Hierarchical task analysis. Taylor & Francis, London

Sutcliffe A, Maiden N, Minocha S, Manuel D (1998) Supporting scenario-based requirements engineering. IEEE Trans Softw Eng 24:1072–1088

Uchitel S, Chatley R, Kramer J, Magee J (2004) System architecture: the context for scenario-based model synthesis. In: Proceedings of the 12th symposium on FSE. ACM, New York, pp 33–42

Van Lamsweerde A (2003) From system goals to software architecture. In: Bernardo M, Inverardi P (eds) Formal methods for software architectures. Springer, Berlin/New York, pp 25–43

Weiser M (1995) The computer for the 21st century. Sci Am 78–89