Architectural Patterns for Context-Aware System

Abdollah Kiani*, Mohsen Amiri Fakhr, Reza Rezaei
Shahid Beheshti University of Iran, Evin, Tehran, Iran.

* Corresponding author. Tel.: +9829903222; email: kian960@yahoo.com
Manuscript submitted January 11, 2015; accepted July 4, 2015.
doi: 10.17706/jsw.10.8.1002-10013

Abstract: Nowadays, context-aware systems are developed by gathering context data and adapting systems behavior accordingly. The purpose of this paper is to examine four architectural pattern context-aware systems in terms of the ability to be implemented. The pattern which can overcome the complexity of these systems and can be more extensible is better to be implemented. This paper is based on a review of Architectural pattern context-aware systems from 2005 to 2009. The patterns include notes on: Example, Problem, Solution, Structure and Benefit. Finally, we will then focus on advantages to be implemented of each pattern.

Key words: Architectural patterns, context-aware systems, quality attributes, software architecture.

1. Introduction

In these days, the use of mobile devices such as smart phones, notebooks and PDAs are becoming increasingly popular. Weiser was the first one (1991) introduced pervasive as a term to seamless integration of devices into the user's daily life. Appliances must make the user and his tasks the central focus rather than computing devices [1].

1.1. Background

As new market opportunities, frameworks, Platforms and technologies become available, systems ought to be reformatted to accommodate them, and frequently this follows large-scale and systematic restructuring [2]. By decoupling users from devices, pervasive computing technology always offers computing. If you want to make satisfactory service for the user's application and services, you should know about their context and automatically adjust to their context-awareness [3].

1.2. Context-Aware System

When one system is able to use context information, it has the feature of Context-awareness. If a system can extract, interpret and use context information and adjust its functionality to the current context of use, it is called context-aware. Context-awareness is a primary issue in emerging fields such as ubiquitous and mobile computing. This computing must be realized by those working in a context-aware system, where it is felt that context is a key in their attempts to distribute computer technology into our lives. In the design of context-aware systems, most challenges must be addressed. These systems can be used in different ways; the application of them refers to special needs and conditions such as the location of sensors (local or remote), the numbers of possible users (one user or several), the available resources of the used appliances (high-end-PCs or small mobile devices) or the facility of more extension of the system [1].

Context-awareness is a main issue in developing fields such as omnipresent and mobile computing [4].

1.3. Software Architecture

The software architecture is the backbone of software system because it is the most important part of the system development. It influences all over the parts of a system, and when the architecture is changed after design or implementation, it has a lot of expenses. Therefore, it is important to determine software architecture carefully. Before the system has been constructed, software architecture gives a basis for analysis of software systems and accordingly, assists to manage risk and decreases expenses during the software development [5].

Important issue is that Context-aware systems are more complex than traditional systems. The important point of context-aware system is that it must be modular to diminish complexity of the system and to improve extensibility. The systems must be user friendly and at the same time they should support dynamic configurability and extensibility. For these needs, an extensible, modular, and dynamically configurable software architecture has been proposed [6].

1.4. Architectural Pattern and Quality Attributes

Architectural Patterns are reusable packages incorporating expert awareness [7]. Patterns for software architectures also display other desirable properties:

1) Patterns offer a common vocabulary and understanding for design principles;
2) They are appropriate means to document for documenting software architectures;
3) They support the construction of software with defined properties;
4) They support constructing the complex and heterogeneous software architectures; and
5) They aid to control the software complexity [8].

Quality attribute requirements are parts of an application's nonfunctional requirements, which capture the many facets of how the functional requirements of an application are achieved. General software quality attributes include scalability, security, performance and reliability [9].

Variability specifies parts of the system and its architecture which stay variable and are not completely defined during design time. Variability permits the development of different versions of an architecture or system [10].

2. Paper Structure

As mentioned earlier, we have introduced patterns include notes on:

Examples: referring to some examples or more extensive overviews of the systems that implement these patterns.

Problem: Focusing on the problems the patterns address. In other words, discussing on characteristics of the application requirements the designers lead to select the patterns.

Solution: The system model captured by the patterns, together with the components, connectors, and control structure that form the patterns.

Structure: A figure illustrating a typical pattern, interpreted to display the components and connectors.

Benefits: the general benefits of using the patterns.

In Section 4, the presented approaches have been discussed, and then, quality attributes such as Reliability and Maintainability to decrease complexity and Scalability, and Adaptability to increase the extensibility of each pattern have been compared.

And, in Section 5, some concluding remarks and presents and some future works in this area have been drawn.

3. Architectural Patterns

We have presented four architectural patterns that can be usable in the development of context-aware systems, including WCAM pattern, Event-Control-Action pattern, Action pattern and architectural pattern for context-based navigation.

3.1. WCAM Pattern

WCAM (Watcher, Controller, Action, and Model) architecture divides a context-aware system into four different and independent constituents. Watcher observes the external environment, and Controller controls the cooperation between Watcher and Model. Model collects contextual information, elucidates the information, and produces contexts for system. A set of services and context adapters is referred to Action. And the sensors that sense values of internal or external environment are referred to Watcher. Controller has Controller Module and Service Descriptor. Controller Module links three constituents: events from Watcher, context from Model, and services in Action. Whether the data are business related or context related, Model is connected to data. Some of the Action or the services are context-aware services, and others are traditional services.

3.1.1. Example

Assume a visitor goes to a museum, he registers his information (background knowledge, language) and his mobile phone information (facilities, phone number), and receives an RF tag. After registration, the visitor walks around the exhibits and then accesses the RF tag to the reader attached to the exhibit. Then the system sends a message to the visitor. The message involves the information about the exhibit, or the URL for the multimedia data about the exhibit. Fig. 1 shows the overall layout of the system.

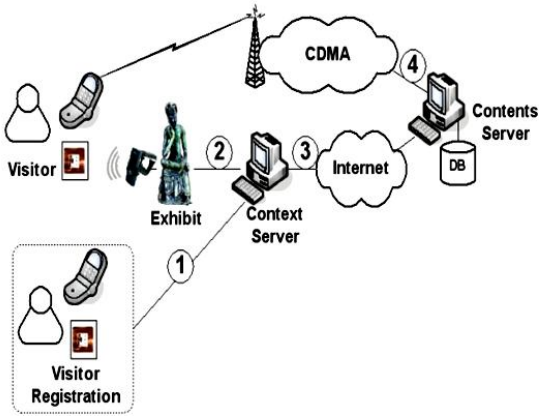

Fig. 1. Service scenario of my guide system [6].

3.1.2. Problem

Reducing the complexity and increasing extensibility are important in a context-aware system.

General requirements for context-aware systems have been mentioned below:

- Distributed System: Most of context-aware systems are normally distributed.
- Heterogeneity: System developers should remember the Context-aware system contains heterogeneous subsystems.
- Mobility: Developers should consider two things, for mobility: functions correlated with mobility and nonfunctional needs for mobility. Mobility is important character of context-aware system.
- Dynamic Adaptability: According to the change of context, a context-aware system should be able to adapt its behaviors.
- Extensibility: Extensibility is a requirement for the context-aware systems to support addition of

new services, new contexts, and new rules for mapping contexts and services.

- Scalability: A couple of external data have been used by Current context-aware systems for their context modeling. Nevertheless, these systems will adopt huge sensor data from sensor networks for their context, in the near Future.
- Configurability: Context-aware systems use user preference and intention as context, and they, closely, connect with users.
- Portability: The client side code is able to run various platforms and devices, so some parts of context-aware systems are correlated with portability.
- Supportability for restructuring: A traditional service plus context-awareness feature creates a context-aware service. In other words, a context aware service is the amalgamation of a traditional service and the context-awareness.

3.1.3. Solution

By decomposing the system into four constituents (watcher, controller, action, and model), the pattern decouples the concerns of context concern and service concern:

- Compatibility and configurability are essential issues in Watcher. That is why; we propose a layered architecture for Sensor Module in Watcher.
- Scalability and dynamic configurability are the important issues in Controller Module. The reason of the importance is that they are rudimentary elements for adding new services and contexts without modification of source. New contexts and services can be added in Model and Action on condition that Controller Module supports scalability and dynamic configurability.
- Compatibility and scalability are the key issues in Model.
- The separation of client side and sever side and supportability of restructuring traditional services are the significant issues in Action.

3.1.4. Structure

Four constituents (watcher, controller, action, and model) construct WCAM architecture that each of them has its own responsibility in a context-aware system. Sensing the value of context element is the role of Watcher. Linking context and services, also, is the role of Controller. Action means a set of services, and model is the part of context management. The four constituents have their own tasks and they do not affect each other. Therefore, we can replace one of them with minimum effect on other constituents.

Fig. 2 illustrates the layout of WCAM architecture. Action consists of two parts: user device and server side. User device is referred to a device that interacts with users. Server-side is referred to a server that supplies services [6].

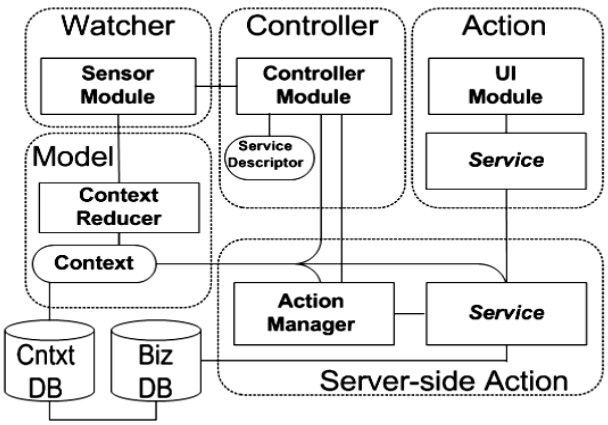

Fig. 2. WCAM architecture structure [6].

3.1.5. Benefits

Reducing complexity by separation of concerns is one of the advantages of WCAM. As a solution of ten requirements, this pattern focuses on reducing complexity by separation of concerns.

Watcher, as one of the components of this pattern, senses external environment and system internal with sensors. By decoupling physical sensor and logical sensor, Watcher mainly centers on compatibility. Scalability and dynamic configurability are the important issues of Controller that bridges contexts and services. Compatibility and scalability are the vital issues of Model that processes context information and deduces context[6].

3.2. Event-Control-Action Pattern

The Event-Control-Action architectural pattern supplies a high level structure for the systems that proactively react upon context changes. In this pattern, the goal is discerning context concerns from reaction (communication and service usage) concerns, under control of an application model. An application model explains the behavior of the application, which may be determined by means of, for instance ,condition rules. This pattern has been planned in a way that context management issues, for example, sensing and processing context, are unrelated to issues concerning reacting upon context changes [8].

3.2.1. Example

An example in the medical domain of this application would be a Tele-monitoring Application, which monitors epileptic patients and supplies medical assistance moments before and during an epileptic seizure. The abilities of this application are measuring heart rate variability and physical activity, predicting future seizures and contacting relatives, healthcare professionals are sent to the patient's current location automatically. The purpose of using this system is to help the patient to have a more routine and safe life.[8]

3.2.2. Problem

The example presented in the last section imposes challenging requirements to the support platform:

- The platform must attempt to collect context information, such as the patient's blood pressure and heart rate in order to predict possible disorders.
- It is required to know the patient's and volunteers' locations, and to take proximity information.
- There is a need to connect with the patient full time.
- The doctor ought to establish a real time streaming connection.
- An ambulance needs to take the patient to the nearest hospital, in the case of a serious situation.

Moreover, it may not be practical to apply such an application within a single business party. In fact, this application is accomplished with the collaboration of several business parties: the location providers, the providers of algorithms to analyze heart rates, the doctor clinic, the hospital, the connectivity providers, and the manufacturers of monitoring devices, among others. There should be an consensus on certain architectural patterns to distribute of the responsibilities among these parties and the coordination of distributed functions [8].

3.2.3. Solution

The purpose of the Event-Control-Action architectural pattern is providing a structural scheme to enable the coordination, configuration and cooperation of distributed functionality within services platforms. Under the control of an application behavior, the pattern divides the tasks of gathering and processing context information from the tasks of triggering action in response to context changes. We presume context aware application behaviors are described in terms of condition rules, the condition part determines the situation under which the actions are enabled. In fact, logical combinations of events represent the

conditions. The occurrence of the actions follows the observation of events, under control of condition rules. One or more Context Processor components shape and observe the events. A Controller component, empowered with condition rules explaining application behaviors, regards the events. When the condition turns true, an Action Performer component starts the actions specified in the condition rules. Actions, such as a SMS delivery or a simple web services call, are operations that influence the application behavior in response to the situation defined in the condition part of the rule. An action can also be a complex composition of services [8].

3.2.4. Structure

Fig. 3 illustrates a class diagram of the Event-Control-Action pattern as it is presumed to be applied in a context-aware services platform.

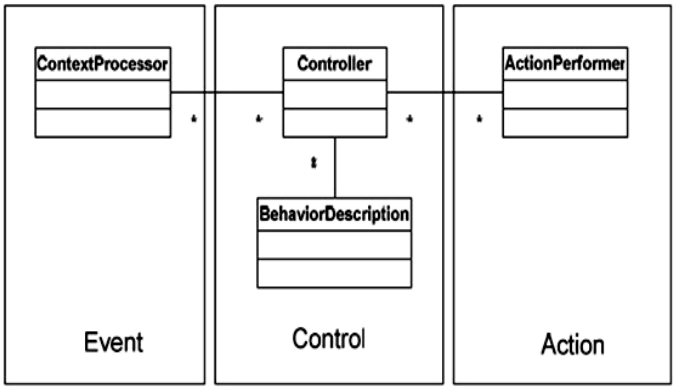

Fig. 3. Event-control-action pattern structure [8].

On the left side of the figure, there are context concerns, which represent the Context Processor component. This component relies on the form and definition of context information. In the middle part of the figure, The Controller component is placed. This component is supplied with application behavior descriptions, represented by the Behavior Description class. Finally, the action concerns are pointed on the right side of the figure. Actions, which can be a service invocation on (external or internal) service providers or a network, can be triggered by The Action Performer component [8].

3.2.5. Benefits

The Event-Control-Action pattern has effectively enabled the dispersal of responsibilities in context-aware services platforms through implementing the classic design principle of separation of concerns.

3.3. Actions Pattern

In order to support designing and implementing action concerns within context-aware services platforms, the Actions architectural pattern supplies a structure of components. The goal of this pattern is decoupling action aims from action implementations and coordinating composition of actions [8].

3.3.1. Example

Assume the example of tele-monitoring, mentioned in section 3.2, in the medical platform. This application monitors epileptic patients and gives medical assistance moments before and during an epileptic seizure. It attempts to send a warning message to the patient, to call his close relatives, to notify volunteers close to the patient of a possible seizure, to send healthcare professionals to the patient's current location when no volunteer is available [8].

3.3.2. Problem

Some of the actions such as sending a warning message to the patient, calling the relatives and notifying nearby volunteers may be executed independently in parallel. Nevertheless, calling healthcare professionals is only enabled in case notifying nearby volunteers has not succeeded; no volunteers are momentarily available. Furthermore, some actions may produce a sequence of other actions. For example, healthcare professionals should know the patient's medical report, to select relevant medication, to check availability of transportation, and so on. In order to manage coordination of actions, the services platform should supply mechanisms, especially when dependencies exist. Moreover, there should be agreements on architectural decisions to distribute and to coordinate of actions in a flexible and decoupled fashion. And the platform should support the independency of an action aim from its execution [8].

3.3.3. Solution

To enable coordination of actions and dividing of action implementations from action aims, the Actions architectural pattern delivers a structural plan. This pattern consist of an Action Resolver component that implements coordination of dependent actions, an Action Provider component that describes action aims and an Action Implementer component that states action executions. An action purpose explains an intention to accomplish a computation with no indications on how and by whom these computations are executed. "Call relatives" or "send a message" are instances of action purposes. Various ways of implementing a given action purpose are defined by the Action Implementers component. For instance, the action "call relatives" may possibly have various implementations, each defined by a telecom provider. And finally, in order to resolve compound actions, which are decomposed into indivisible units of action purposes (indivisible from the platform standpoint) the Action Resolver component applies some techniques [8].

3.3.4. Structure

Fig. 4 illustrates a class diagram of the Actions pattern as it is presumed to be applied for context-aware services platforms. Both the Action Resolver and Action Provider components inherit the features of the Action Performer component, and hence are both enabled to accomplish actions. The Action Resolver component implements compound actions fragmenting them into indivisible action purposes, which are further executed separately by the Action Provider component. Action Providers may be communication service providers or (application) service providers. Communication service providers accomplish communication services, such as a network request, while service providers accomplish general application-oriented services, implemented either internal or external to the platform, for example an epileptic alarm generation or an SMS delivery, respectively [8].

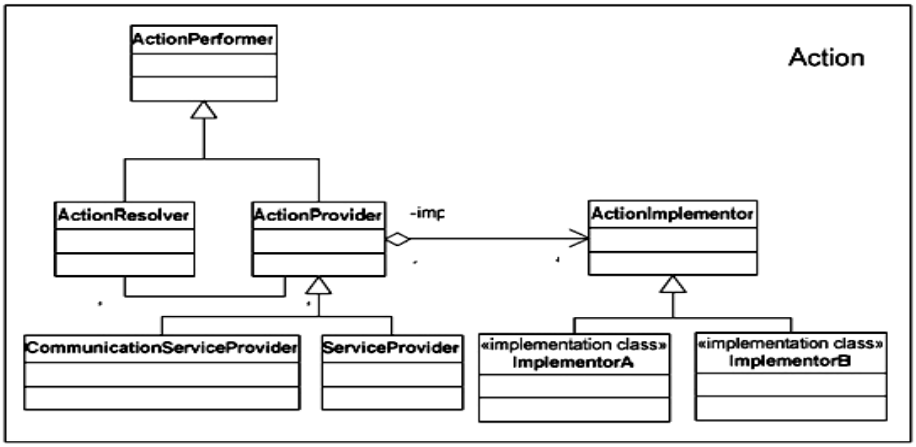

Fig. 4. Action pattern structure [8].

3.3.5. Benefits

By defining a structure of Action Resolvers, Providers and implementers, the Actions pattern has enabled the coordination of compound actions and the separation of abstract action purpose from its implementations. This effort avoids permanent binding between an action purpose and its implementations, permitting the selection of different implementations at platform run-time. Moreover, abstract action purposes and concrete action implementations may be changed and extended individually, improving dynamic configuration and extensibility of the services platform.

3.4. Context-Based Navigation Pattern

This pattern aims at integrating contextual information in applications serving mobile user navigation in physical environments. In this pattern, context is defined as the information about the user and the environment, beyond the map.

3.4.1. Example

Imagine a college or university campus in which many new students begin their academic job in a new and unknown location. It is clear that an application supplying campus navigation would be useful for the students. The student would benefit from extra information surrounding the campus, such as personalized class information, event information and any campus police announcements. Most users are grateful to be informed on social media aspects, such as a friend's location. But how to integrate such contextual information into the campus navigation process is controversial [11].

3.4.2. Problem

Navigation is influenced by many factors. As it is mentioned, the issue is to integrate context into the navigation process. In detail, structuring the information necessary to provide context-aware navigation in a generic way and supplying an architecture for context-based navigation are problematic. The following items can assist to find a possible solution:

- Adaptability: the system must be able to accommodate an influx of spontaneous events because the events can happen at any time.
- Functionality: The system must supply a means for integrating navigation data, depending on the surrounding context.
- Scalability: regarding to the number of users, map size, and context size, the system must be scalable.
- Extensibility: if new context is added, the architecture of the system must be extensible.
- Usability: The system must be used friendly and usable in various situations.
- Efficiency: The system responds to contextual changes in a timely manner[11].

3.4.3. Solution

For the context-aware navigation, Fig. 5 includes a block diagram of the architecture, containing several core components:

- In order to get directions to some destination location, mobile users use a wireless terminal and a client navigation application.
- User context consist of user information, current location, destination, and the path. On the other hand, user information contains authorization level, preferences of the user, and social media context (e.g. friends' location). A sensor (e.g. GPS for outside location) or a localization service (e.g. for indoor localization) supplies Current location.
- In order to define the navigable trajectories for the user, the map includes one or more graphical layers of the navigation environment (e.g. satellite picture) and a graph data structure (e.g. vertices and segments).

- A policy can be explained as a set of rules detailing a set of tasks to be implemented when a set of conditions are satisfied.
- External events symbolize contextual information that have timing and location, and that are pertinent to the navigation process, e.g. traffic events, accidents, parades, road work, etc.
- Tasks are feasible commands set out by the policy engine, such as user notification (e.g. a nearby road accident, a friend location in the proximity) and rerouting (e.g. request to recalculate the path due to a recent event on the current trajectory) [11].

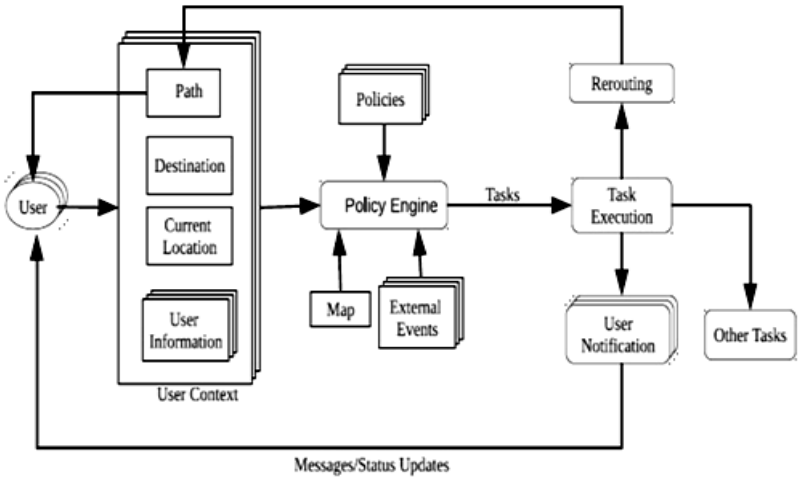

Fig. 5. Block diagram of the architecture for context-aware navigation [11].

3.4.4. Structure

In Fig. 6, some concepts named policy rules, context elements, and the relationships between them are illustrated. The diagram contains categories with behavior related to policy processing (policies, rules, tasks) and also subcategories of Context Element that are instantiated in memory at runtime from an XML or Database store using a query engine.

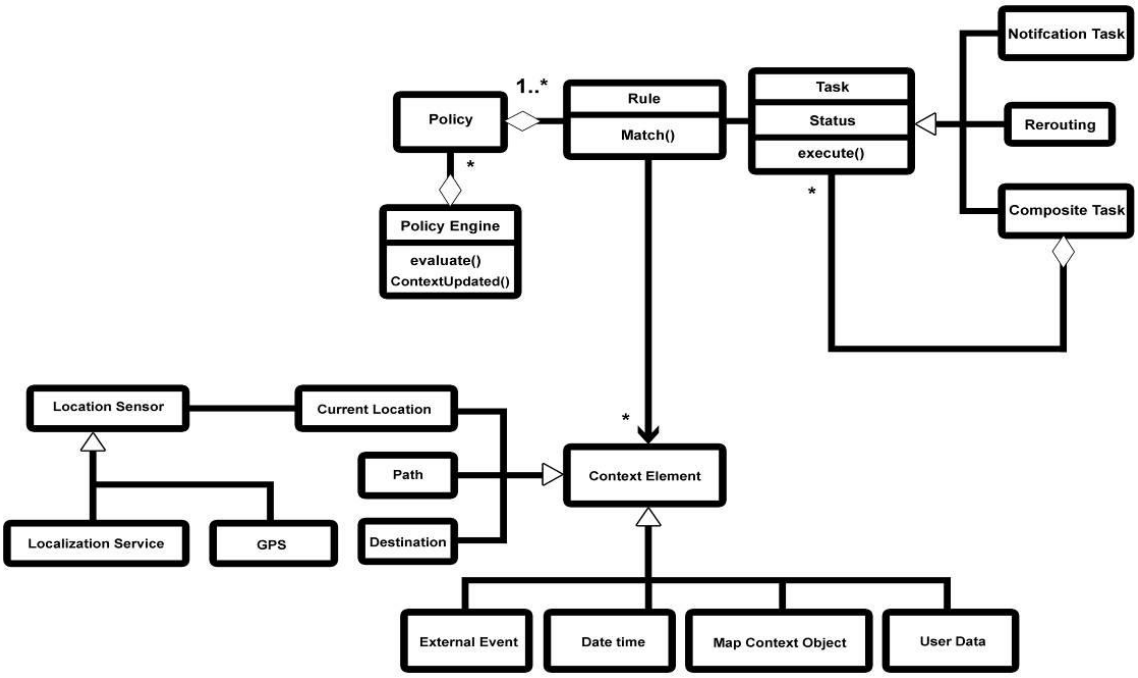

Fig. 6. Context and policy concepts class diagram [11].

3.4.5. Benefit

The profits of the context-based navigation pattern can be:

- Adaptability: in order to the addition of new policies (new features) without having to change and rebuild the system, firstly, the separation of policies and the policy engine should permit.
- Functionality: The policy engine authorizes integration of user's current location, intended destination, and contextual information to make a trajectory path. When the trajectory path is updated, new events can be reported and accounted.
- Scalability: task implementation and policy matching may be divided between the clients and a variety of distributed services running on multiple servers. The client must act even when disconnected from the network. A reduced set of the user context could be kept on the client side.
- Extensibility: new policies can be added at any time without altering the system, because the policy engine and the policy specifications are generic.
- Usability: The system is able to react to changes affecting the user and is user friendly. For example, it sends user notifications about the traffic conditions and other events.
- Efficiency: The system can also reply to changes affecting the user in near real-time[11].

4. Discussion of Approaches

Table 1. The Main Aspects Complexity of the Patterns

Pattern Name	Complexity	
	Reliability	Maintainability
WCAM	•	•
Event-Control-Action	•	-
Action	-	•
Context-based navigation	•	-

In Table 1, Reliability and Maintainability, two important available quality attributes of the discussed patterns have been compared. These attributes reduce complexity.

Table 2. The Main Aspects Extensibility of the Patterns

Pattern Name	Extensibility	
	Scalability	Adaptability
WCAM	•	•
Event-Control-Action	•	-
Action	•	•
Context-based navigation	•	•

In Table 2, two other important available quality attributes of the presented patterns, Scalability and Adaptability, have been compared. These attributes increase extensibility. A context-aware pattern must decrease its complexity and increase extensibility. We can diminish complexity and increase extensibility with an emphasis on general software quality attributes contains scalability, performance and compatibility.

Scalability and dynamic configurability is more essential quality attributes to add new contexts without modification of the source.

The most important available quality attributes in each pattern:

- WCAM pattern focuses on compatibility, extensibility, scalability, and dynamic configurability.
- The Event-Control-Action pattern centers on responsibility for supplying and retrieving contexts and supportability of restructuring traditional services.
- Actions pattern focuses on dynamic configuration and extensibility.
- Context-based navigation Pattern centers on Extensibility, Scalability, compatibility, Functionality, Adaptability.

5. Conclusions

In this study, four architectural patterns have been presented that can be beneficially applied in the development of context-aware systems, namely the WCAM pattern, the Event-Control-Action pattern, the Action pattern and the navigation pattern. Quality attributes as : (1) Extensibility and flexibility in the Event-Control-Action pattern, (2) Compatibility and Adaptability in The navigation pattern, (3) Scalability and compatibility in the WCAM pattern and (4) improving extensibility and flexibility the action pattern assisted us managing the heterogeneity and complexity of the awareness infrastructure, which can be developed and maintained by various business parties.

Further study topics should include:

- Manage frequent notifications from the server in a rich and dynamic contextual environment may drain the battery.
- The risk of faulty user notifications must be mitigated with a reputation or reward system.
- Focusing on variability of one specific quality attributes in architectural pattern.
- However, architectural patterns are easy to learn and useful to apply in the real environment sentence.

References

[1] Baldauf, M., Dustdar, S. & Rosenberg, F. (2007). A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, *2(4)*, 263-277.

[2] Barnes, J. M., Garlan, D., & Schmerl, B. (2014). Evolution styles: Foundations and models for software architecture evolution. *Software and Systems Modeling*, *13(2)*, 649-678.

[3] Hong, J. Y., Suh, E. H., & Kim, S. J. (2009). Context-aware systems: A literature review and classification. *Expert Systems with Applications*, *36(4)*, 8509-8522.

[4] Salvaneschi, G., Ghezzi, C., & Pradell, M. (2012). Context-oriented programming: A software engineering perspective. *Journal of Systems and Software*, *85(8)*, 1801-1817.

[5] That, M. T. T., *et al*. (2014). Preserving architectural pattern composition information through explicit merging operators. *Future Generation Computer Systems*.

[6] Choi, J. (2008). Software architecture for extensible context-aware systems. *Convergence and Hybrid Information Technology*.

[7] Yoshioka, N., Washizaki, H., & Maruyama, K. (2008). A survey on security patterns. *Progress in Informatics*, *5(5)*, 35-47.

[8] Costa, P. D., Pires, L. F., & Sinderen, M. V. (2005). Architectural patterns for context-aware services platforms.

[9] Gorton, I. (2006). *Essential Software Architecture*. Springer Science & Business Media.

[10] Mahdavi, H. S., *et al*. (2013). Variability in quality attributes of service-based software systems: A

systematic literature review. *Information and Software Technology*, *55(2)*, 320-343..

[11] Cardei, M., Jones, B., & Raviv, D. (2013). A pattern for context-aware navigation. *Proceedings of the 20th Conference on Pattern Languages of Programs*.

Abdollah kiani is a M. Eng. student in e-Learning Computer Science Department, Shahid Beheshti University of Iran. His research interests are software architecture pattern, health information system, and context-aware system. He is a lecturer at the Department of Paramedical, Faculty of Health Information Technology, Lorstan University of Medical Sciences, Iran.

Mohsen Amiri Fakhr is an M. Eng. student in e-Learning Computer Science Department, Shahid Beheshti University of Iran. He is now an engineer at the Seyed Shoada Hospital Tehran, Iran. His research interests are hospital information system, cloud computing.

Reza. Rezaei is a Ph.D. of software engineering, Faculty of Computer Science and Information Technology, University of Malaya. His research interests are interoperability, cloud computing, and service oriented architecture. He is a lecturer at the Department of Computer Engineering, Faculty of Technical and Engineering Sciences, Islamic Azad University of Saveh Branch, Iran.