
Technical Report: C3-516_01

Mahmoud Hussein, Jun Han, and Alan Colman
Faculty of Information and Communication Technologies (FICT)
Swinburne University of Technology, Australia

July 2010

1 This research was supported by the Australian Cooperative Research Centre for Advanced Automotive Technology (AutoCRC).
Table of contents

TABLE OF CONTENTS ......................................................................................................................... 1

1 INTRODUCTION ............................................................................................................................. 2

2 MOTIVATING SCENARIO .................................................................................................................. 3

3 RELATED WORK ............................................................................................................................. 5

4 CONTEXT AWARE ADAPTIVE SOFTWARE SYSTEMS REQUIREMENTS ........................................ 6

4.1 REQUIREMENTS FOR THE CONTEXT MODELING ................................................................. 6

4.2 REQUIREMENTS FOR THE SYSTEM RUNTIME ADAPTIVITY .......................................... 8

4.3 REQUIREMENTS CONCERNING THE SYSTEM-CONTEXT RELATIONSHIPS ........................... 9

5 CASE STUDIES ............................................................................................................................... 11

5.1 FRAMEWORKS FOR CONTEXT-AWARE ADAPTIVE SYSTEMS ........................................... 11

5.2 FRAMEWORKS FOR ADAPTIVE SYSTEMS ......................................................................... 14

5.3 FRAMEWORKS FOR CONTEXT-AWARE SYSTEMS ............................................................. 18

5.4 SUMMARY .................................................................................................................................. 20

6 FUTURE CHALLENGES .................................................................................................................. 22

6.1 THE SYSTEM CONTEXT MODELING .................................................................................. 22

6.2 THE SYSTEM RUNTIME ADAPTIVITY ............................................................................... 22

6.3 THE SYSTEM-CONTEXT RELATIONSHIPS ......................................................................... 23

7 CONCLUSION ................................................................................................................................. 24

REFERENCES ..................................................................................................................................... 24
Faculty of Information and Communication Technologies
Swinburne University of Technology


Mahmoud Hussein, Jun Han, and Alan Colman

Abstract

Self-adaptivity and context-awareness have been proposed to provide the ability for a software system to adapt itself at runtime to cope with changes in its environment and the user needs. In research, self-adaptivity is more concerned with how to adapt the system, while context-awareness is more concerned with how to model, process, and manage the system environment information. In general, context-aware adaptive software systems need to consider both aspects and have the ability to sense the context changes and adapt upon them. However, research in self-adaptivity and context-awareness has been carried out largely in separate communities, with limited reference to each other. In this report we present an analytical survey of research into context-aware adaptive software systems from both of these perspectives, with a particular focus on issues concerning their integration. We identify the requirements of integrating context-awareness and self-adaptivity to better understand their relationship, analyse the current research in context-awareness and self-adaptivity regarding these requirements, and distil the research challenges in developing context-aware adaptive software systems.

Keywords: Self-adaptivity, context-awareness, context and system modeling, software architecture.

1 Introduction

There is an increasing demand for software systems that dynamically adapt their behavior at run-time in response to changes in their users’ preferences, requirements, and context and the systems’ operational environments and underlying infrastructure [1-2]. Changes can also be induced by failures or unavailability of parts of a software system itself. In these circumstances, it is necessary for a software system to change its structure and/or behavior as necessary to continue achieving its new and existing goals. Dynamic changes that occur while the system is in operation require the system adaptation to occur at run-time. As the system evolves at runtime, this poses challenging question about how to specify, design, verify and realize this system [3].
Research into systems that adapt themselves in response to context and/or requirements changes has been conducted by researchers from two communities with two different emphases: self-adaptivity [1, 4] and context-awareness [5]. On the one hand, research in context-awareness is more concerned with how to model, process, and manage the context information. But limited on how a system adapts itself in response to unanticipated changes in the context information. Furthermore, it is usually not concerned about requirements changes while the system is in operation. On the other hand, research in self-adaptivity is more about how to adapt the system’s structure and/or behavior in response to requirements and/or context changes (possibly unanticipated changes), with less attention to how context is modeled, processed, and managed. In practice, the line between the two is rather blurred, and the engineering of a context-aware adaptive system needs to consider both aspects in a holistic manner. With the objective to gain a better understanding of the relationship between context-awareness and self-adaptivity to advance the research and practice in this area, we in this report identify the requirements of considering context-awareness and self-adaptivity in an integrated manner, analyze the current research in context-awareness and self-adaptivity relative to these requirements, and distill the research challenges in developing context-aware adaptive systems.

2 Motivating Scenario

Context-aware vehicle systems are examples of context-aware adaptive software systems, which need to (a) model, process, and manage their environment information, and (b) adapt themselves according to the changes in their environments, and the driver/passenger needs. We present below a number of scenarios where such a vehicle systems are used. They will serve to (1) help the discussion of the report; (2) show the need to consider both context-awareness and self-adaptivity aspects in developing context-aware adaptive vehicle software systems.

James’s vehicle is equipped with context-aware software applications to navigate and organize his affairs while on the road. The vehicle/applications’ connectivity with the outside world (including other vehicles, smart road infrastructure, services over the internet etc.) is enabled by built-in Bluetooth, DSRC, 3G and GPS technologies. The applications in his vehicle include the Vehicle-Navigation-Assistant (VNA), which assists James with route planning, detects/avoids congestions, and searches for and communicates with other vehicles, road infrastructure, and external commercial services that match the driver’s preferences such as services that provide (a) locations of the pharmacies and (b) the traffic information. Below are a few scenes, he experiences when driving home one day.

Scene 1: The trip starts when James entering his vehicle. The vehicle detects James and activates his profile. The vehicle starts to play his favorite music, select his default route, and display a message, “you need to get the medication,” that has been specified in his diary beforehand.

Scene 2: James needs to re-plan his way home to stop at a pharmacy. The vehicle system (1) loads the VNA application to search for and communicate with the best available services that provide the pharmacies locations, and the traffic information, while considering his preferences (e.g. lowest service price, and etc); (2) shows the available routes for James based on the current context (i.e. James’s preferences, vehicle location, pharmacy location, and traffic information), and then he chooses the suitable one and starts his journey home.

Scene 3: While he was driving, (a) the traffic information service provider become unavailable, and then the VNA finds another traffic information service provider and communicates with it; (b) the road side units send the local area speed limit to the VNA, and then the vehicle speed limit is adjusted; (c)
rain is detected by some dedicated sensors of his vehicle. He is informed of the speed limit, and he changes the vehicle speed accordingly. In addition, James decided to use the adaptive cruise control (ACC), which used to maintain a preset vehicle speed and automatically adjust the vehicle speed to keep a specified distance with the preceding vehicle, while considering the rain level (i.e. the ACC operational behavior in rainy conditions). When the rain stopped, the ACC is returned to its normal operational behavior; (d) congestion happened in his route, and this information is received from the traffic information service provider. Then, the VNA shows to James the available routes, and then he chooses the suitable one and continues his journey home.

The above scenario shows that the context-aware adaptive vehicle system needs to:

(1) **Model, process, and manage** its context information, where (a) the system’s environment has a large amount of information about the driver, the vehicle, and the vehicle environment (e.g. the nearby vehicles, the services providers, the road side units, and etc.), which affects the system operation and then a separate model of the context information need to be maintained to reduce the system complexity; (b) the raw context information are required (e.g. the front vehicle speed and location), but higher level context information is more powerful (e.g. the possibility of colliding with the front vehicle) and needed by the system. Therefore, the raw context data need to be processed to infer high level context information; (c) during the system operation the context information are changeable where there is a context value change (e.g. congestion information changes), or a context type change (e.g. the set of context information needed by the system is changeable from a situation to another). In addition, these changes can be anticipated at design time or unanticipated which it will be known at runtime. Therefore, the system needs to have an explicit runtime representation of its context to (a) enable the context model adaptation and evolution at runtime; (b) reduce the system complexity.

(2) **Have the ability to adapt itself** in response to changes in (a) the driver/passengers needs (i.e. the system requirements), and (b) the system environment, the system resources, and/or the system itself. **First**, in response to user needs/requirements changes the system adapts itself. For example, the driver can request (a) the route planning, and then the VNA is loaded (i.e. structure change); (b) the change of the desired distance gap with the preceding vehicle (i.e. parameter change) (c) a new application for diagnosis assistant (i.e. unanticipated requirements change) and then it is added to the system’s structure. **Second**, the system needs to detect environment changes and act upon them. For example, (a) if the driver specified in his diary to get the medication. Then the system is changed, where the VNA searches for and communicates with a service that provide the location of the nearest pharmacy (i.e. structure change); (b) the vehicle route is changed in response to congestion being detected (i.e. parameter changes); (c) the ACC behavior is different from the normal operating conditions to rainy condition (i.e. behavior change); (d) if a new smart road side units is added which provides new context information (i.e. unanticipated context change) or the context information goes outside its expected range, which was not considered during the system design, then the system should take the required adaptation actions to cope with this change. **Third**, the failure, unavailability, or performance degradation of the running system component(s) needs a response from the system by (a) replacing the running component in case of failure or unavailability; (b) adding new component(s) to improve the system performance. For example, if the traffic information service provider becomes unavailable, then a new service provider is selected and used. Finally, the system needs to be aware of its resources and adapt itself in response to changes in it. For example, when the system needs to use a route planning service, it needs to know is this service is available or not (i.e. the system needs to maintain a resources/solution context model). In addition, when there is a traffic information service provider that has a better connection and/or service quality than the
Currently connected one, the adaptation is triggered to switch between the two service providers. Therefore, the system needs to have ability to adapt itself for coping with anticipated and unanticipated context/requirements changes.

(3) Capture the relationship between the system and its environment explicitly, where the system needs to detect context changes and act upon them. These context information can be (a) operational context, which is needed by the functional system to continue its operation (e.g. the vehicle location is needed by the route planning to operate correctly); (b) management context, which cause the system to adapt form a state (structure and/or behavior) to another (e.g. the unavailability of the driver preferences cause the system to adapt from a route planning algorithm to another). For fast and efficient reaction of the system to context changes, it needs to have an adaptable system-context relationships, where in current context situation only small set of rules are use to decide the required adaptation action, and then the decision making time is reduced (i.e. decision maker efficiency). In addition to cope with unanticipated context changes, the system-context relationships need to be changed at runtime (i.e. adding/removing/replacing new adaptive behavior). Furthermore, the representation of the system-context relationships should not increase the system complexity and the developer errors by hardwiring the relationships with the functional system. Therefore, the system context relationships need to be represented explicitly and be able to be changed at runtime to (a) have efficient decision making mechanism; (b) cope with unanticipated changes; (c) reduce the system complexity and development errors.

3 Related Work

Several surveys from different viewpoints have been carried out such as surveys on context modelling techniques, runtime system adaptivity, context aware systems, and self-adaptive systems. But In general, these surveys have not systematically considered the requirements for the integration of the context-awareness and self-adaptivity in developing the context aware adaptive software systems. In particular, they did not consider our novel proposed requirements as discussed briefly in the next paragraphs and with details in the next section.

From the context modelling perspective Bettini et al. [6], Bolchini et al. [7], and Strang et al. [8] surveyed and evaluated the proposed context modelling techniques. These authors have proposed different requirements for modeling the context as a separate entity from the system, without pay attention of how this context information affects the system. Therefore, we concentrate on the context modeling requirements from the perspective of how the context affects the system operation and adaptation in developing context-aware adaptive software systems. In addition to the traditional context modeling requirements, we present new requirements that were not considered in current context modeling surveys such as (1) the social context model and the context model subjectivity as context modeling aspects; (2) the context model (a) runtime representation; (b) future trends predication, (c) adaptation and evolution as context management operations.

From the runtime system adaptivity perspective, several authors discussed the requirements for the traditional [9-11] and adaptive [12-14] system modelling, but our concern is the adaptive system modelling aspects in relation to its context (i.e. system reaction to context changes). In particular how the system multiple models are managed during the software execution from the system-context relationships perspective. Therefore, we presented system’s solution context, runtime models, and runtime evolution as new requirements that were not considered in the current surveys.
From the system-context relationships perspective, Matthias et al [5] identify the requirements that context aware systems should have, and then an evaluation for research in this field is provided, but they don’t consider the requirements that are posed by the context into the system (i.e. the system-context relationships), where this type of research is more concerned with how to modeled, process, and manage the system context information. Research in self-adaptivity is more concerned with how the system responses to its context and requirements change without pay attention to how the context is represented and processed inside the system. Therefore, Salehie et al. [2], Huebscher et al. [15], Nami et al. [16], and Bradbury et al. [12] have surveyed the proposed self-adaptive software systems, but they don’t consider the requirements for integrating the context awareness with self-adaptivity. Therefore, we present here our new requirements that are related to the system-context relationships such as (a) explicit relationships representation; (b) runtime relationships adaptation and evolution.

4 Context Aware Adaptive Software Systems Requirements

The above scenario shows that there is a need for integrating the context-awareness and self-adaptivity aspects in developing context-aware adaptive systems. To integrate both aspects, we need to maintain the context-awareness and the self-adaptivity perspectives together with their relationships.

While the context-awareness needs to model and process the information about system environment to be further used by the system. Therefore, the context modeling requirements need to be considered from the system-context relationship perspective (Requirements 3.1). On the other hand, self-adaptivity need to have a system that can cope with the context/requirements changes (both anticipated and unanticipated), and then the system need to be designed with adaptation in mind (Requirements 3.2). Furthermore, the requirements for the mechanism that integrates the context-awareness and self-adaptivity (i.e. the system-context relationships) need to be considered (Requirements 3.3).

4.1 Requirements for the Context Modeling

The term context-aware appeared initially in [17], where the application is aware of context like locations and identities of nearby people and objects. Later, one of the most widely accepted definitions is given in [18], considering context as “any information that can be used to characterize the situation of entities (i.e., whether a person, place or object) that are considered relevant to the interaction between a user and an application, including the user and the application themselves.” In this definition they consider context relevant to interactions between the user and the system, but these interactions is not only between the system and its user, but also with other systems in its context (e.g. the vehicle system relationship with the service providers). Furthermore, in case of two systems are interacting, the interactions themselves need to be modeled as a part of the context, because they affects how the systems are going to operate in relation to each other (e.g. interactions between two vehicles in a cooperative convoy). From this idea, a definition of context that is more related with our target is the following “Context is any information that is relevant to the interactions between the system and its context entities, including the context entities, the system, and their interactions themselves.”

4.1.1 Context modeling aspects:

Physical and situational context: For a correct system operation, it needs to know certain facts about its physical environment. For example the vehicle system needs to know the vehicle’s fuel level and...
location, where in case of low fuel level the system communicates with a petrol station locator service, to find the nearest petrol station relative to the vehicle current location. Furthermore, the physical context information are required (e.g. the front vehicle speed, and location), but higher level context information is more powerful (e.g. the possibility of colliding with this vehicle) and needed by the applications. This information (i.e. situational context) is the aggregation of multiple pieces of physical context information of one or more context entities to infer high level information, with respect to an entity that is interested in this situation. For example, the driver is interested in harsh weather conditions, which can be inferred from the temperature value, the rain status and the wind speed. Therefore, it is important that the context model is able to represent (enable inferring) such high level situational information with respect to an actor interest.

**Relationships:** Not only there is need to represent the information about the context entities, but also their relationships need to be modeled which affects the system operation. Therefore, the context model should enable the representation of (1) physical context entities relationships, where there are various relationships between the physical context entities that affects the system operation [19] such as (a) nearby relationship between two vehicles; (b) engaged in relationship between the person and an activity (e.g. the driver is engaged in route planning); (c) depend on, where a change to the value of one property (e.g. rain detected) may impact the values of other properties (e.g. the vehicle front window wiper status); (2) interaction-oriented/social context, where a set of rules, obligations and understandings that influence an individual’s action with respect to a group in a particular situation are modeled for making the system cooperate with other interacting systems (e.g. the service providers) [20]. Furthermore, this social context can be between the context computational entities themselves that is relevant to the system (e.g. the interactions between the driver and the vehicle, where he can put the seatbelt on).

**Subjectivity:** In the interaction oriented context (i.e. social context), individual actors may have their own perception of the interaction/relationship, and then the context can be modeled from the viewpoint of one of these actors (e.g. one of the vehicles in the cooperative convoy) or from the domain where they interact in (e.g. the traffic management system) [20]. Likewise, the physical or situational context can be modeled from the perspectives of a user, an application, or the organization the user/applications are in.

**Information quality:** Context information can be of varying quality, depending on its source [21-22]. The information quality affects the system adaptation decision, where incorrect information leads to unwanted adaptation actions. Therefore, the context information model should support quality indications.

### 4.1.2 Context Model management

**Explicit runtime representation:** While the system is in operation, it needs information about its context (e.g. sensed context such as the vehicle speed or stored context such as the driver diary), and then this information need to be captured and represented at runtime. Furthermore, the context information should be represented as first class entities and not be embedded into the application code to (1) reduce the application complexity, and (2) make the system able to have deliberative strategies for responding to context changes.

**Runtime context model adaptation:** The context model is consisted of a set of entities and in large systems the number of context entities can be large. In addition during, the runtime not all the context entities are required based on the system current state (i.e. current configuration and/or behavior).
Therefore, the context model should have the ability to be adapted at runtime to reduce the monitoring overhead.

*Runtime context model evolution*: As the system and its environment will evolve, so the context model should have the ability to evolve to cope with unanticipated environment changes. Therefore, context model should be flexible and extensible, where new context elements shall be allowed to enter the system environment and represented in the context model.

*Context history and predication*: The current and stored context information can be used to predicate the future trends of the context values. For example, based on the previous stored traffic congestion information, the system can predicted the future congestions before their occurrence. Therefore, there is a need to store the context information for future context values prediction.

### 4.2 Requirements for the System Runtime Adaptivity

During the system runtime (i.e. the system execution), it needs to adapt itself in response to context changes. These changes can be anticipated during the design time, or unanticipated till the runtime. For the any system to achieve the runtime adaptivity, it needs to consider the following properties:

*Runtime system’s structure and behavior models*: In response to context change the system adapt its structure (i.e. re-configuration) and/or behavior to cope with this context change while preserving its high level goals at runtime. Therefore, a runtime representation of the system’s structure and behavior needs to be maintained for (1) providing the running system state to the system management to take the correct actions; (2) supporting the system adaptation realizations by applying the adaptation actions to the system models and then to the running system.

*Solution context*: During the runtime, the system structural adaption is performed by removing, adding, or replacing a system component(s). To perform the addition and replacement adaptation actions, the system need be aware of the currently available components (i.e. the solution context) that can be used [23]. Furthermore, to change the system parameters (e.g. the number of used units of the server memory); there is a need to know the permitted parameters changes (e.g. the server memory usage can only be increased by 5%). Therefore, a model for the system’s available resources should be maintained at runtime.

*Runtime quality models*: Adapting the system behavior during the run time in response to context change (i.e. switching from one configuration/behavior to another) affects the system quality attributes such as (1) *response time*, where there is small response time to stop the vehicle in case of an obstacle appeared. Therefore, when the adaptation action is to replace the obstacle detection algorithm, the specified required response time should be considered; (2) *availability*, where the system components have a failure (unavailability) probability and affects the overall system availability such as the navigation service provider availability affects the vehicle navigation availability. Therefore, based on the required availability level of the navigation application, the service provider is selected. Furthermore, there are other quality attributes that need to be maintained such as security and so forth. These quality attributes should be preserved before, during, and after performing the adaptation. Therefore, the non-functional requirements models need to be maintained at runtime.

*Runtime system adaptation*: During the software system execution it needs to adapt itself in response to the changes into its context. Therefore, the system structure and its behavior need to have the ability to be adapted at runtime for handling the new situations and make the system survive for a
long time. In addition, changing the system structure and its behavior models at runtime means that any change in one view should be reflected in the other views to keep them consistent.

**Runtime system evolution:** The system will have a long life time. During this life time the system need to adapt itself in response to new context change (i.e. unanticipated changes that was not considered at the design time) while it is in operation. Therefore, the system structure and its behavior need to have the ability to cope with unanticipated context changes without stopping the system.

**Runtime reasoning:** To build a correct context-aware adaptive system, its models should have a formal representation that enables the validation of the system structure and behavior against its global invariants (i.e. the system constraints) in design time or during the runtime when the system is changed in response to its context change. Furthermore, this formal representation can be used to check the consistency between the context model and the system model to initiate the adaptation process in case of consistency violation [23].

### 4.3 Requirements concerning the System-Context Relationships

The system needs to adapt itself in response to its context and/or requirements changes. Here we more concerned with the context change, where we assume the requirements are not changeable and need to be preserved when the system is adapted during the runtime.

The adaptive systems are coming in many different forms. But, they have common method for adapting the system in response to context changes, which called the adaptation loop [1-2, 24]. This loop is used to (1) detect the changes that need a response from the system (i.e. monitoring and analyzing); (2) decide the required adaptation actions (i.e. deciding). These actions can be for regulating the system parameters or changing the system’s structure and/or behavior; (3) apply the adaptation actions to the running system (i.e. acting). But, when the system and its context are coming to play inside this loop, it is changed a little bit based on the context information type, where each context type has different relationship with the system as follows:

1. **Operational context**, which is required by the system to continue its operations (e.g. to find the nearest pharmacy, the vehicle location is needed), and this context type needs only the monitoring/analyzing operations as the loop that is shown in the top part of Figure 1 (i.e. from the context to monitoring and analyzing operations and then to the functional system).

![Figure 1: The system-context Relationships](image)
To capture the relationship between the system and its context, there is a need for a suitable method for representing the system-context relationships requirements and enable their realization as follows:

**Explicit relationships representation:** The system and its context are not separate entities, and the context changes affect the system (i.e. the operational and management context) and vice versa where on request from the system the needed context elements (i.e. the context awareness model) are changed. The system-context relationships will be complex in large scale system, where there are a large number of adaptations, which corresponds to the large number of environment changes, and the developer will face difficulties in considering them. Therefore, the relationships between the system and its context need to explicitly represented at modeling stage to (a) ease the task of the system developer; (b) reduce the errors that can happen by considering the system-context relationships implicitly during the implementation phase.

**Relationships realization and correctness:** In large scale systems designing the system-context relationships (in particular the adaptation logic) are error prone and writing the code corresponding to it is complex. Therefore, the relationships should be easily transformed to code that represents them. In addition, it should enable the correctness check by verifying the system adaptive behavior.

**Relationships runtime adaptation:** Adaptation decision must be performed within specific time duration and no more than that, where late adaptation actions will not have any value (e.g. in safety/mission critical adaptive systems) [24]. Furthermore, the decision making that takes long time affects the system performance. Therefore, the system needs to decide the required adaptation actions to satisfy the new context as fast as possible with less overhead by having adaptable system-context relationships that can be activated or deactivated at runtime.

**Relationships runtime evolution:** For the system to cope with unanticipated context changes (i.e. new context information to be considered), the system-context relationships need to be evolved. Therefore, these relationships should be adaptable at the runtime by allowing new relationships to be defined at runtime or existing relationships can be modified or removed.

**Automated relationships runtime evolution:** Adaptive system may be deployed in unexpected environment, where the communication with the system developers is infrequent [25]. Therefore, the adaptation decision must be performed automatically without (or with little) human guidance where the system drives automatically the new adaptation actions to cope with unexpected environment changes (i.e. driving new system-context relationships automatically). Furthermore, generating on-the-fly adaptation actions may or may not achieve the intended goals, so it need to be stored for using or avoid using it in the future.
5 Case studies

Over the past decade a large number of researchers have been concerned with the research in the context-awareness, and the self-adaptivity. Therefore, in the following sub-sections we briefly overview a number of frameworks and analyze them with respect to the requirements we identified in the previous section. We classified these frameworks into three categories based on the degree of how they treat the context-awareness and self-adaptivity.

5.1 Frameworks for Context-Aware Adaptive Systems

Recently we have seen some research efforts that start to consider with the same degree both context-awareness and self-adaptivity [6, 23, 26-30]. However, there is still a long way to go to fully address the requirements we have identified (see Table 1).

Rainbow framework provides mechanisms for (1) monitoring the system and its context to be reflected in their models; (2) performing the analysis of these models to initiate the adaptation process; (3) selecting the required adaptation strategy; (4) effecting the needed changes to the running system [23]. They use Acme to model the system and its context [31]. The system context is represented as a set of resources (i.e. the solution context) that support the adaptation process. Furthermore, modeling the environment as part of the Acme architectural model enables the runtime representation of the context model. But, the Acme model is not suitable to (1) represent the interaction oriented context and the physical context entities relationships; (2) deal with incorrect and imprecise context information where they assume the system probes provide correct information about the system and its environment. They model the system structure to enable its construction and this model is maintained at runtime for performing model reasoning to initiate the adaptation process in response to context change. Their context model analysis is performed by checking model constraints violations. In large scale systems, where there are large number of model constraints and model elements, this checking process takes long time and affects the adaptation mechanism efficiency. They use a language called Stitch to capture the relationship between the system and its environment changes, but modeling the adaptive behavior manually using the Stitch language is a problem in large systems where many variations need to be captured.

MUSIC project [26, 32] is a component based framework that is used to optimize a system’s overall utility in response to environment changes. They have a quality of service model, which describes the system composition together with the relevant QoS dimensions and how they are affected when the system is going to change from one configuration to another. This quality of service model is used for selecting the new configuration with the best utility to cope with environment changes. They propose an ontology-based environment model for modeling the context information. This model facilitates the model driven development of context aware adaptive systems. Their context model is concerned with modeling the real world that cause the system to adapt as a response to any change in it (i.e. the management context), with less attention to modeling the system resources that support the adaptation process and the operational context that is needed by the system components. Subjectivity is supported where the context can be modeled from the application's perspective or from the user perspective. The set of observable context elements can be changed at runtime to achieve optimal monitoring. Their approach is based on ontology which support the reasoning process, but ontology based techniques have a bad performance in reasoning complex situations [6]. Their system model is based on a system component meta-model representing the system structure. It can be used to design a specific system model that conforms to it, and then the system code can be
generated. They capture the relationship between the system and its context through the context values that are defined in the utility function of each component. But designing utility functions may be problematic for large systems that have complex system behavior (i.e. complex utility functions).

Ayed et al. [27, 33] extended the UML class diagram to incorporate the definition of context information that is relevant to the system. They add the following stereotypes to the UML class and sequence diagrams: (1) context, for defining the simple context elements such as location and the bandwidth, but it is not suitable to model the interaction oriented context; (2) collection process, for collecting the required context information and can be event based or periodic monitoring; (3) context quality, for indicating the quality of the received context information; (4) context state, which specifies the context combinations (i.e. high level context) that needs a response from the system. Another extension is for both class and sequence diagrams to incorporate the system adaptive structure and behavior, where they add (1) variable component structure, which defines several variants to the same component or component optional functions that can be added or removed; (2) variable architecture, in which some components are defined as optional and can be added/replaced/removed when needed; (3) variable sequence, which specifies different sequences in the sequence diagram (i.e. sequence variant) for each possible system behavior. They model the relationship between the system and its context by associating a context state with the variable sequence, architectural, or structure to specify in which context state this variant is activated. This method (1) adds complexity to the system where the context model is intertwined with the system model, (2) limits the ability to cope with unanticipated environment changes where the relationship should be specified at design time, and (3) has the difficulty in modeling large system where there is a large number of variations to be considered.

André et al. proposed a generic framework for the adaptation of web services [28]. They use probes (i.e. hardware and/or software sensors) that are distributed through the system to collect information about the system and its environment. The context data can be received as events or monitored periodically, and then it passed to the monitors to infer high level information from these low level context data. The collect context information is passed directly to the adaptation manager without having an explicit context model at runtime. Therefore, their technique is not able to (1) cope with unanticipated environment changes; (2) represent the interaction oriented context and the context information quality. The system is constructed as a set of service that are composed and changed during the runtime without having an architectural model of the system. Furthermore, they don’t maintain the system behaviour or the non-functional requirements models at the runtime. For capturing the relationship between the context change and the system adaptation, they use (1) action policies, where adaptation actions are specified at design time to have a fast system reaction; (2) utility policies, where the specific adaptation actions are not specified until the runtime.

Andrade et al. proposed an approach to cope with unanticipated context changes by separating the adaptation logic from the core system artefacts (i.e. the functional system), and then these autonomic artefacts can be changed at runtime [34]. The proposed platform have three main modules (1) environment, that provide interfaces that need to be implement for monitoring specific environment elements. This module contains only the needed monitors by the system to reduce the monitoring overhead. This method directly monitors the system environment and the gathered information is passed to the adaptation rules, and then the framework is unable to represent the interaction oriented context, simple context entities relationships, and the context information quality; (2) adaptation, which contains a set of adaptation action polices in form of a tree that specifies the required adaptation actions based on the environment state. Each rule contains two parts: the adaptation rule
as an expression based on the monitored device, and actions that represents the required architectural change. These policies are constructed as a component based to enable dynamic change at runtime; (3) redeployment, which is responsible for applying the required adaptation actions to the running system. It contains two methods: (a) handle adaptation to perform pre-deployment operations for ensuring safe adaptation actions and (b) perform deployment to apply the required change. They model the system structure and enable its change without pay attention to the system behaviour change. Furthermore, the architectural model doesn’t have a formal representation to perform the required reasoning for correctness and models consistency.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Context Modeling and Management</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Physical and situational context</td>
<td>+</td>
<td>+</td>
<td>~</td>
<td>+</td>
<td>~</td>
<td>~</td>
<td></td>
</tr>
<tr>
<td>Relationships</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td></td>
</tr>
<tr>
<td>Subjectivity</td>
<td>-</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td></td>
</tr>
<tr>
<td>Information Quality</td>
<td>-</td>
<td>-</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td></td>
</tr>
<tr>
<td>Runtime Representation</td>
<td>+</td>
<td>+</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td></td>
</tr>
<tr>
<td>Runtime Adaptation</td>
<td>+</td>
<td>+</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td></td>
</tr>
<tr>
<td>Runtime Evolution</td>
<td>-</td>
<td>-</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td></td>
</tr>
<tr>
<td>System Runtime Adaptivity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System's Structure and Behavior</td>
<td>S</td>
<td>S</td>
<td>S/B</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>Solution Context</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Quality Models</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Runtime Adaptation</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>~</td>
<td></td>
</tr>
<tr>
<td>Runtime Evolution and Reasoning</td>
<td>-</td>
<td>-</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td></td>
</tr>
<tr>
<td>System-context Relationships</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Context Type</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>Explicit Relationships</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Relationships Realization</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>Relationships Adaptation</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Relationships Evolution (Automated)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>~</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

CAPucine is an approach for adapting the systems at runtime in response to context changes using the dynamic software product line [29]. Their approach is based on two processes for product derivation: (1) a process for composing different assets to generate the product (initial phase), and (2) a process that introduce context-aware asset at runtime to specify the dynamic adaptation. This asset is consisted of a set of rules that defines what are the required actions in response to context change. Therefore, during the runtime this asset is used to get the environment state and decide the needed change from a software product to another. The second process (i.e. context aware asset) is corresponding to system-context relationship. They consider the context implicitly inside the context.
aware asset and this limits the system ability to (a) incorporate unanticipated context information and the system reactions to them; (b) have adaptable context-awareness to reduce monitoring overhead. Furthermore, the system-context relationships cannot be changed at runtime.

Scatter is an approach for selecting over-the-air product variant for a mobile device based on its characteristic (i.e. device context information) [30]. They use feature models to represent (1) the system functionality and, (2) the mobile device features. They address the relationship between the mobile device feature changes (i.e. device context model), and the system change (i.e. the system feature model) by associating references between the two models by identifying their dependencies. They only consider the device context, without having an explicit context model for the system environment (i.e. it is unable to perform context management operations). In addition, the context entities should be defined during the design time and linked with the system and then their approach cannot cope with unanticipated context changes. Furthermore, the adaptation is performed during the deployment time and not during the software execution (i.e. at runtime).

5.2 Frameworks for Adaptive Systems

There have been quite a number of research efforts investigating self-adaptivity in response to context/requirements changes [3, 35-38] (see Table 2).

Heaven et al. have developed an approach to building self-adaptive systems, where the system adapts its structure in response to environment change while preserving its high level goals [35, 39]. They use Labeled Transition Systems (LTS) to model the system domain [40]. This model captures the states the system and its environment can be in, and the environment changes are the actions that move the system from one state to another. In their technique, the context information is embedded in the system domain model. As such, it is not suitable to model the interactions oriented context. They do not consider the solution context either. As the LTS is built based on the system and environment variables, the LTS for a large system has a large number of variables describing the system and its environment, and is difficult to build and validate. Furthermore, they have a technique to safely apply the adaptation action where changes are applied to the system model and then reflected in the running system while preserving the system state consistency.

An approach was introduced by Zhang et al. for creating formal models for the self-adaptive system behavior [3]. In their approach the system’s adaptive behavior is separated from the no-adaptive behavior. This separation makes the system models easier to specify and verify. They use Petri-Nets to model the system’s adaptive behavior, where they use context change as guidance for the transition between the system states [41]. As such, they implicitly consider the context as part of the system’s adaptive behavior model. This technique in modeling the system environment is not rich enough to capture the interaction oriented context, the solution context, or the context information quality. Furthermore, there is not an explicit context model maintained at runtime. Therefore the context management operations cannot be performed. Furthermore, they concentrate on the system behavior change with less attention to the system structure change. Regarding the relationship between the system and its context, they define how the system can safely move from one state to another in response to context change using the Petri Nets transition operation. But, modeling the adaptive behavior in a state based model is difficult for large systems, because the number of system states grows exponentially.

Morin et al. have proposed a technique to handle the exponential growth of the number of configurations that are derived from the system variability [36, 42]. They combine model driven and
aspect oriented approaches for coping with the complexity of the adaptive software systems. Their context model represents the environment variables which are used in the adaptation rules (i.e. the management context). They did not have a separate context model, and the context information is modeled within the system model. They do not consider the solution context, the interaction oriented context, or the context data quality. Furthermore, they do not have a runtime representation of the context model. As such, it cannot be modified at runtime to cope with unanticipated environment changes. Their structure model is based on a meta-model which has a runtime representation and can be used to validate the system invariants. They capture the relationship between the system and its context using the adaptation rules where context changes are evaluated to identify system structure changes and guide the selection of the system features.

StarMX has been proposed to ease the task of realizing self-adaptive systems based on java [37], where this system realization have many challenges that posed by the system complexity and its adaptive behaviour. Their framework is based on (1) a rule engine for the analysis and the decision making operations; (2) JMX technology for the sensing and effecting purposes. StarMX architecture is consisted of two main elements: (1) execution engine, which contains the self-adaptive operations as processes and each one is as an execution chain that define how to perform the required operation. Each process (e.g. effecting) is linked with specific system elements that provide the needed function (e.g. a specific effectors that created by the system developer) via an anchor object; (2) set of services, which are used to enable the execution of the of adaptation loop processes such as (a) lookup service to access the anchor objects; (b) activation mechanism for triggering the adaptation process and this triggering can be event based or timer based; (c) caching service for improving the performance by holding a reference to the anchor object. They discussed the process of engineering a system with the self-adaptive properties using their framework as follows: (1) define the system non-functional requirements such as security, reliability, and so forth; (2) define the set of resource that can be used by the adaptation logic such as sensors and effectors; (3) define the system adaptation polices; (4) configure the StarMX framework by linking what was defined in the previous steps with the running functional system. Their main concern is how to develop the system management logic without paying attention to how the system and its context are modelled. Furthermore, they don’t maintain the system behaviour models at runtime to cope with system behaviour change.

PLASTIC project aims to support the adaptation of web services based on the current context or QoS change [38], to assure that the users get the best quality of service. They have a service conceptual model that is implemented as UML profile and supports the service life cycle from the design to implementation and validation and then the execution. Furthermore, they have the model of the service quality attributes (e.g. performance, reliability, and etc.) and the service behaviour model. The environment elements, which they are concerned with, are the available resources such as bandwidth, CPU time, and memory (i.e. the execution context). Therefore, they monitor these parameters directly without having an explicit context model. But, considering the system environment internally limits its ability to (1) represent the interaction-oriented context, context entities simple relationships, context quality; (2) enable context model adaptation and evolution. They capture the system-context relationships by adding annotations to the system variants, where each variant has his resource demand (i.e. execution context), and then based on the user context and the request quality of service, the suitable variant is chosen. Their approach is limited to the adaptation at the deployment time only (i.e. service discovery), therefore the monitoring and acting in the running system is not required. Furthermore, the system runtime adaptations and/or evolutions are not supported.
Folch et al. proposed a technique to use the architectural models for achieving the runtime adaptability [43]. Their architecture models consisted of the system basic component and for each component (a) a set of alternative variants are defined; (b) a utility function is defined to evaluate the different component variants in response to context change. When there is a context change, the architecture variants are evaluated to select the best suitable variant (i.e. the one with the highest utility). Their context model contains only the needed information for the utility functions. They do not specify what the type of their context model is, and it seems that they use a simple model for the needed attributes and their values. Therefore, their context model is not rich enough to satisfy our context modeling aspects and the management requirements. They capture the relationship between the system architecture changes and its context by the values that are defined in the utility function of each component to compare between different variants. The developer should design the utility function, which it is a problem in large system where there are complex utility functions.

Oreizy et al. proposed an architecture-based approach for runtime software adaptation [44]. Their infrastructure supports two simultaneous processes for the building of self-adaptive software systems: (1) system adaptation management, as the process of detecting, analyzing, and planning for adaptation in response to the system (i.e. internal) or its environment (i.e. external) changes; (2) system adaptation management, as the process of applying the changes to the running system. They observe the system to know its internal changes that cause the adaptation, but they assumes the external observations are provided to the system via the system operator or the other interacting system without paying attention to how it can be observed. Therefore, they don’t maintain the system environment model, which means that (1) the interaction oriented and high level context information cannot be represent; (2) there is no runtime representation of the context model to cope with unanticipated environment changes. Their approach is architecture-based where an explicit architectural model is maintained at runtime and the changes to it are reflected into the running system for ensuring that the model and the running system are consistent with each another. They use adaptation action policies to capture the relationship between the context change (i.e. observation) and the system adaptation actions (i.e. the system response) as knowledge based expert system [25]. Their approach faces the difficulty of writing adaptation polices for large system, where the developer needs to specify all the observations and their corresponding actions.

Medvidovic et al. proposed an extension to the C2 architectural style [45]. Their architectural style supports the programming-in-the-many (PitM) requirements such as self-awareness, distribution, heterogeneity, dynamism, and mobility. The C2 style supports the top and bottom connectors, and then they extend it by adding the side connector to enable system components interactions. Furthermore, they introduced a border connector to enable the system interactions with the other systems (i.e. the interaction oriented context). They have a second architecture level called meta-level and it is connected with the running system, which maintains a representation of it. They assume that the correct system state is reflected into the runtime model and then they do not address how to assure the correctness of the context information. They adapt the system in response to the failure or the unavailability of the system components (i.e. internal changes) without pay attention to the external world. Therefore, they do not have an explicit environment model which limits the system ability to cope with unanticipated environment changes. The relationship between system state change (i.e. failure or unavailability of components) and the system response is described using action policy, where a set of actions are specified such are adding, removing, and replacing system components or connectors. Furthermore, they do not maintain a behavior model, where they are not concerned with adapting the system behavior.
Table 2: Comparison between current adaptive systems relative to our requirements

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Approaches</th>
</tr>
</thead>
<tbody>
<tr>
<td>Context Modeling and Management</td>
<td></td>
</tr>
<tr>
<td>Physical and situational context</td>
<td>Heaven et al [35]</td>
</tr>
<tr>
<td>Relationships</td>
<td>Zhang et al [3]</td>
</tr>
<tr>
<td>Subjectivity</td>
<td>Morin et al. [42]</td>
</tr>
<tr>
<td>Information Quality</td>
<td>StarMX [37]</td>
</tr>
<tr>
<td>Runtime Representation</td>
<td>PLASTIC [38]</td>
</tr>
<tr>
<td>Runtime Adaptation</td>
<td>Folch et al. [43]</td>
</tr>
<tr>
<td>Runtime Evolution</td>
<td>Oreizy et al. [44]</td>
</tr>
<tr>
<td>CAWAR</td>
<td>PitM [45]</td>
</tr>
<tr>
<td>CASA</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>S B S S S</td>
<td>S S S S S</td>
<td>S S S S</td>
</tr>
</tbody>
</table>

| System Runtime Adaptivity                       |                                |
| System’s Structure and Behavior                 |                                |
| Solution Context                                |                                |
| Quality Models                                  |                                |
| Runtime Adaptation                              |                                |
| Runtime Evolution and Reasoning                 |                                |

| System-context Relationships                     |                                |
| Context Type                                    | M M M M M M M M M M M M M M |
| Explicit Relationships                          |                                |
| Relationships Realization                       |                                |
| Relationships Adaptation                        |                                |
| Relationships Evolution (Automated)             |                                |

(–) Unsupported    (~) Partially supported    (+) Supported    (M) Management context    (O) Operational context    (S) Structure model    (B) Behavior model

CAWAR framework is service based approach [46], which is based on a context adaptation meta-model. This meta-model has four types of services: (1) context element, that has information about the context entity to be monitored; (2) sensors, which are used to acquire the context data; (3) interpreter, that are used for processing the gathered context data to obtain high level information; (4) actuators, that are responsible for deploying the required adaptation actions. This meta-model can be used to have a representation of the system context (i.e. the context model) or the system itself (i.e. the system model) as context elements. But, this model is not suitable to model the interaction oriented context. They use another service which called model activator that (1) implements the adaptation mechanism as an action policy, and (2) supports the inference about the system and its context models to initiate the adaptation process. For coping with anticipated environment changes (i.e. new context element), a new system configuration is designed and the adaptation logic is changed to link the context change with this new configuration. But, in large systems where there are a large number of adaptations it is not possible to model all the possible variations using their technique.

CASA is a contract based adaptive software architecture framework that is proposed to adapt the system in response to changes into its resources (i.e. limit resources, resources with different reliability, or new resources are added) [47]. The system is design to support adaptation by having a
set of components that are active (i.e. the running system) and another set of inactive components to be used for the system reconfiguration (i.e. the solution context). The system contract (i.e. adaptation policies) specifies the system as a set of zones, which each zone has its provided quality level. Furthermore, each zone have different configuration with different resource requirements. Therefore, the system can adapt from one configuration to another in the same zone based on the available resources while maintaining the system quality requirements. This technique of modeling the relationship between the context change (i.e. the resources changes) and system adaptation (1) is not suitable in large systems that have large number of context variations with their corresponding configurations; (2) limits the system ability to adapt the adaptation logic at runtime, where it should be specified at design time. They monitor the system resources, and then their status is passed to the adaptation policies to select the new configuration. Therefore, the interaction-oriented context, physical context entities relationships and the context information quality indication cannot be represented where they do not have a context model.

5.3 Frameworks for Context-Aware Systems

Several middleware and architectures have been proposed to develop context-aware software systems [19, 21, 48-49]. They are primarily concerned with how to model, process, and manage the context information as shown in Table 3.

The Service-Oriented Context-Aware Middleware (SOCAM) project introduces architecture for building context-aware systems [19]. They use a central server for gathering the context information from distributed context providers and then this information is processed for the use by the system applications. The context-aware applications are the top layer of the architecture which uses the different levels of context (i.e. from raw data to high level context). The applications’ behavior is adapted according to any change in their context. They propose a two layer ontology-based context model for representing the environment information: (1) a general layer captures the general context entity, for example person, place, etc, and (2) a specific layer extends the general layer for a specific domain (e.g. driver, vehicle, etc.). These two layers enable loading and unloading of specific domains (e.g. moving from home to vehicle and to the office). To capture the relationships between the context change and the system reaction, they use a set of predefined rules that relate the responses of the system to context changes (i.e. operational context).

The space of observable entities of context aware systems may be huge and each application may have specific monitoring requirements. For such situations, CA3M [48] uses a model-driven engineering approach to help the designers to capture context-awareness concerns in context models according to a context-awareness meta-model. Their context awareness model has a runtime representation and is used to monitor only the environment entities needed by the application. Therefore, they are concerned with how to monitor the environment and provide its information to the system application without paying much attention to how the system is going to react to context change. In addition, they address the system-context relationships by defining a set of contracts [50]: (a) **observation contract**, which used to specify the required quality of context by the application; (b) **notification contract**, which specifies when the context has a specific value it should be notified to the application; (c) **adaptation contract**, which is used to define the required adaptation actions in specific context situations. But, they do not specify how these actions are going to be applied to the system where they do not represent the system management explicitly.

Sheng et al. proposed an approach to developing context-aware web services. Their development environment (ContextServ) [49] relies on a UML meta-model called ContextUML [51]. This meta-
model is based on four concepts. First, context elements represent the web service environment and can be simple element (i.e. one context element) or composite element (i.e. the aggregation of multiple simple elements). Their modeling does not provide the ability to capture the interaction oriented context or the subjectivity aspect. Furthermore, they have a runtime representation of the context model. But it is not adaptable to handle unanticipated context changes. Second, context sources elements specify what the context entities are. They are used to specify the monitored context entities for each service separately, and to switch from one context provider to another based on the provided context quality at the runtime. Third, service modeling elements specify the service operations and their input and output messages. The system can be viewed as a collection of services that are represented in one model. But this model is not represented at runtime to enable structure change. Furthermore, they do not maintain a system behavior model. Fourth, context awareness elements specify the services and their context relationships and are classified into two types: (a) context binding, where a service needs the context attribute value as parameter; (b) context triggering, where in response to a context change an action is taken to adjust the service operations.

Henricksen et al. present a software framework that addresses the challenges of building context aware systems [21]. Their aim is to simplify the design and the implementation of these systems. The framework is based on (1) a context modeling approach that describe context information together with the user preferences, and (2) a pair of complementary programming models (branching and triggering models) for capturing the relationship between the context changes and the application reaction. They formalize the context model by concentrating on some aspects like information quality and temporal aspects of contexts. They enhanced the context model by making it able to be represented at runtime using XML and can be shared between multiple applications. Their model is not able to capture the interaction oriented context and can not be used to handle unanticipated context change where the context model is not adaptable. As the SOCAM project [19], they use the adaptation rules for addressing the relationship between the system and its context. Furthermore, the system structure/behavior models are not maintained, where their main concern is to model and process the context information.

ScudWare is a semantic and adaptive middleware platform for smart vehicle space [52]. They used multi-agent, context awareness, and adaptive component management techniques to construct their middleware. They use ontology as a context model, but their ontology based context model is simple which fails to capture (1) context information quality; (2) interaction oriented context; (3) solution context. Furthermore, they use the context model at conceptual level without having a runtime representation to cope with unanticipated environment changes. The system is composed of multiple agents where they are grouped together as request by the application and there is no explicit system model to be verified. The relationship between the system and its context is not shown, but from their discussion it seems that they use rules to capture this relationship.

A centralized approach that is designed for context-aware mobile applications is a project called Context-Awareness Sub-Structure (CASS) [53]. The project middleware contains (1) SensorListener which, listens for updates from sensors that are located on distributed computers called sensor nodes and then the collected information is stored in a database; (2) ContextRetriever is responsible for retrieving the stored context information; (3) ChangeListener is a component with communication capabilities that allows a mobile computer to listen for notification of context change events. Gaia project aims at supporting the development and execution of portable applications for active spaces [54]. Gaia has services to query and utilize existing resources to enable accessing and using current context, and it provides a framework to develop context sensitive mobile applications.
### Table 3: Comparison between current context-aware systems relative to our requirements

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Context Modeling and Management</strong></td>
<td>Physical and situational context</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td><strong>Relationships</strong></td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td><strong>Subjectivity</strong></td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td><strong>Information Quality</strong></td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td><strong>Runtime Representation</strong></td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td><strong>Runtime Adaptation</strong></td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td><strong>Runtime Evolution</strong></td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td><strong>System Runtime Adaptivity</strong></td>
<td>System’s Structure and Behavior</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td><strong>Solution Context</strong></td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td><strong>Quality Models</strong></td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td><strong>Runtime Adaptation</strong></td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td><strong>Runtime Evolution and Reasoning</strong></td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td><strong>System-context Relationships</strong></td>
<td>Context Type</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td><strong>Explicit Relationships</strong></td>
<td>~</td>
<td>~</td>
<td>+</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td><strong>Relationships Realization</strong></td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td><strong>Relationships Adaptation</strong></td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td><strong>Relationships Evolution (Automated)</strong></td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
</tbody>
</table>

(-) Unsupported  (-) Partially supported  (++) Supported  (M) Management context  (O) Operational context  (S) Structure model  (B) Behavior model

#### 5.4 Summary

Tables 1, 2, and 3 summarize how the frameworks discussed in this section satisfy the requirements identified in the previous section. From these frameworks analysis, we can make several observations as follows:

First, current context modeling techniques consider the physical context without pay regard to the interaction oriented context [20, 45]. Furthermore from the system perspective, there are three context types (i.e. operational, management, and solution). But context-aware systems consider only the operational context [18, 53], and self-adaptive systems consider only management context with less attention to both operational and the solution context [3, 26, 35]. However, the context-aware adaptive system needs to (1) model its physical and interaction oriented context and (2) consider the operational, the tactical, and the solution context.

Second, research in self-adaptivity models the system behavior for verification [3, 35] or the system structure for enabling its construction [23, 26, 42]. On the other hand, the research in context-awareness concentrates on modeling the system context without pay attention for modeling the
system structure or its behavior [14, 19, 21]. However, the context-aware adaptive systems need to model both system structure and behavior for later adaptation and evolution at runtime.

Third, there are three methods for the monitoring operation: (1) continuous, where the components give their status every period of time continuously; (2) event based, where in context change (event), a notification is sent to the system about this change; (3) event/continuous, where the previous two techniques can be used interchangeably. Most of the proposed techniques use the continuous monitoring that provides a correct context/system state, but it has a large overhead. On the other hand, one technique has used event/notification where the correctness of the system/context state cannot be guaranteed [36]. The future direction is to use the combination of them by making a tradeoff between the monitoring overhead and the correctness of the system/context state [19, 49].

Forth, current adaptation mechanisms that are used to cope with context changes (system-management context relationship) are classified into three different categories: (1) rule based, which takes the form of “IF (condition), THEN (actions)”, these rules are pre-computed which the designer should specify [34, 42], (2) goal based, which goals are defined and used to infer the required adaptations actions rather than specifying what to do in current state like rule based mechanism [3, 35]; (3) utility based, which generalizes the goal based adaptation mechanism by quantifying each possible state with a value instead of classifying the states as desired next state or not (binary classification, i.e. 0 or 1) [26, 43]. But, there are recent approaches combine two of the above techniques, which give an efficient adaptation mechanism [23]. Therefore, the future trend is to combine two adaptations mechanisms for getting better efficiency.

Fifth, predicting the future trends of the system context can help in performing proactive adaptation that reduces the system downtime. Therefore, the system context history needs to be stored and then used to predict the context future trends. But most of the recently introduced frameworks do not maintain context history, and even the frameworks that maintain the context history do not perform predication for the context future trends.

Sixth, current research captures the system environment in two ways. The first way captures the system environment implicitly, where environment information is considered as a part of the adaptation technique (e.g. rules, strategies or finite automate) [3, 35, 42, 47]. In these techniques the system context is monitored and the environment values are passed to the adaptation manager to take the required adaptation action. These techniques are not efficient in modeling complex environments where the space of observable entities is huge. Furthermore, they are unable to handle unanticipated environment changes while the system is in operation. The second way captures the system environment explicitly, where an explicit environment model is maintained [19, 21, 43, 50, 52]. Maintaining an explicit context model has a number of benefits. First, it can support the capture of solution context. Second, it supports the initiation of the adaptation process, where the consistency between the system and the context models is checked and the adaptation process starts in case of consistency violations. Third, it reduces the complexity of context and system modeling/validation, comparing to the use of implicit context models. Fourth, it can readily support environment evolution at runtime. Finally, it enables the reasoning process and the storing of this information for future use or predicting the context trends. Therefore, the system should have an explicit representation of its context at runtime.

Seventh, during the software execution the system non-functional requirements need to be preserved and then the system quality models need to be maintained at runtime. But most of the introduced
frameworks neglect this aspect, except limited works that considers the quality attributes implicitly within the system implementation [23, 26].

Finally, engineering the software system with the adaptation mechanisms improves their ability to survive for a long time, but it affects the system performance and resources. Therefore, the adaptation technique effects into the system should be calculated. But current frameworks do not compute this effect, and then the adaptation benefits may be lost by having a system that has high cost or bad performance.

6 Future challenges

In the previous section, we have analyzed the current research in context-awareness and self-adaptivity to assess how they have satisfied the requirements identified earlier. On this basis, we in this section identify the future research challenges in terms of context modeling, system runtime adaptivity, and the system-context relationships. We also refer to the research efforts that have started to address these challenges to ground our discussion.

6.1 The System Context Modeling

The system needs to interact with its context entities (e.g. the nearby vehicle in the cooperative convoy or the service provider that provides the pharmacy locator service). These interactions affect the system operation. Therefore, modelling these interactions is important to ensure that the system is operating correctly [20]. In addition, modelling the interactions can be with respect to one of the interacting entities or the domain where they are interacting in. Therefore the modelling techniques should support the subjectivity in representing the context information. Furthermore, the system needs to be aware of the available resources to take correct adaptation actions [23]. Therefore, the context model needs to represent the solution context.

In order to monitor the system and its context, their entities must provide the ability to be monitored by making available their status. Most of the proposed techniques use the continuous monitoring, which has a large overhead, but the system may operate on low power devices that have limited resources and at any time the system do not need to monitor all environment elements. Therefore an efficient monitoring mechanism with adaptable context-awareness (i.e. changeable set of monitoring elements based on the system situation) is needed to reduce the monitoring overhead.

There have been a number of research efforts (in particular, research that uses an explicit context model) addressing the context model management [19, 21, 43, 50, 52]. However, most of these efforts do not fully address our requirements (see Table 1) such as context model adaptation and evolution during the runtime.

Therefore, a rich context model is needed to (1) capture all aspects of the system context (e.g. the physical, the situational, and the social/interaction-oriented context), and (2) have a runtime representation that can be changed while the system is in operation to cope with unanticipated environment changes and supports context management operations.

6.2 The System Runtime Adaptivity

The system needs to adapt its structure and/or behaviour in response to context changes. Therefore, there is a need to model the system structure and its behaviour and maintain them during runtime. In addition, they should be adaptable to cope with context and/or requirements changes. Furthermore, in
case of maintaining multiple system models, they should be consistent with each other, where any change in one model needs to be reflected in the others.

In response to the context change, the system adapts itself. This adaptation should be performed during the runtime while preserving the system goals, both functional and non-functional (qualities) [35, 38, 55]. Therefore, both the functional and non-functional requirements need to be maintained at runtime. In addition, when adaptations are applied to the system models, these models need to be checked against the system requirements to ensure that the adaptation actions do not violate these requirements.

Not only is there a need to have a consistent system structure, behaviour and quality models, but also there is other needs to improve its usability and effectiveness such as tool support for design-time and runtime activities [11].

Adaptation actions need to be performed at runtime to keep the system in operation as much as possible. As such, online dynamic execution of operations is required rather than offline maintenance or version control operations [55]. This poses a challenge of how to safely apply the adaptation actions at runtime to cope with environment changes. Furthermore, the system will be constructed from components, each of which in turn is a set of components. These components may be distributed so how to apply changes in a decentralized manner is yet another challenge.

The system will be deployed in an environment, which is not totally anticipated during the system design time. Therefore, the functional system and its management should have the ability to be adapted at runtime to cope with unanticipated context changes (i.e. runtime system evolution).

6.3 The System-Context Relationships

The number of adaptations is based on the size of the context variables and their variations. In a large scale system with a complex environment, there are a huge number of possible adaptations. It is difficult to consider a large number of system variations at design time as well as runtime. Therefore, how to consider a large number of adaptation behaviors need to be addressed in modeling the system-context relationship. There are suggestions to solve this problem by providing an automatic or semi-automatic online planner to generate new configurations on-the-fly at runtime [4]. First of all, how practical and generally applicable such an approach is debatable. In addition, to ensure the correctness of the system when this new configuration is applied, runtime verification is needed. This poses the challenge of how to perform fast automated runtime verification.

Adaptation rules are expressive and easy to write, but are prone to error (e.g. rules conflict) [34, 42]. On the other hand building LTS, PN or utility functions is difficult but can be verified for correctness [3, 23]. Therefore, an approach that easily supports both development and verification is required.

In vehicle/mobile systems, the context changes are frequent and may be every second and may need quick adaptive responses from the system. In these systems not all adaptation actions have the same importance (e.g. safety versus navigation based adaptations in a vehicle system). Therefore, priority should be considered in selecting the adaptation actions [56]. In addition, when there are multiple adaptation actions pending, a further question is how to stop a running adaptation action to perform another one with a higher priority and needing immediate attention. In case of stopping adaptation, rollback its changes is required to ensure system consistency. Furthermore, some systems may operate in low powered devices. Consequently, a balance between acting overhead and relevant benefits needs to be considered.
Context-aware adaptive systems may be deployed on low powered devices and then the system performance needs to be considered. A possible way to improve the performance is to have adaptable system-context relationships to reduce the decision making mechanism overhead. Therefore, the system-context relationships (in particular the adaptation rules) should have the ability to be adapted at runtime.

Both the context model and the functional system need to be evolved at runtime to incorporate new context entities, and then the system-context relationships need to evolve. This evolution should be performed while the system in operation to keep the system in operation as much as possible.

7 Conclusion

Context-awareness and self-adaptivity are treated separately in existing research. But real world demands their consideration in an integrated manner as shown in our motivating scenario. In this report, we have presented our vision for context-aware adaptive software systems from the perspective of system-context relationships. We started by setting up the requirements of integrating context-awareness and self-adaptivity in context-aware adaptive software systems. These requirements are partitioned into three categories: (1) context modelling and management, (2) the system runtime adaptivity and (3) system-context relationships, to provide the basis for analysing context-aware adaptive systems and discussing their main research challenges.

Based on this analysis, we have drawn a number of observations such as (1) the system should have an explicit representation of its context model at runtime; (2) most of current context modelling techniques considers only the physical and situational context, however there is also a need to model both interaction oriented and solution context; (3) there is a future trend to use the combination of continuous and event based monitoring (i.e. adaptive monitoring); (4) the future direction is to combine two adaptation mechanisms to gain better efficiency.

The main challenge(s) in (1) context modelling is how to have a context modelling technique that captures all the system environment aspects and supports later adaptation and evolution during the software execution; (2) system runtime adaptivity is to maintain multiple system models (i.e. the structure, behaviour and quality models) and their consistency at runtime and enable system runtime evolution; (3) considering the system-context relationships are how to have an adaptation mechanism that handles a large number of adaptations and can cope with unanticipated environment changes.

To provide a context-aware adaptive software system, solutions to these challenges need to be developed and integrated to provide a comprehensive solution, and supported by an appropriate infrastructure. The long term objective of our research is to develop solutions for these challenges and apply them to systems like context-aware adaptive vehicles system for validation.

References


[40] R. van Glabbeek, "Labelled Transition Systems."


