A System-Context Relationship Oriented Survey of Context-Aware Adaptive Systems

Technical Report: C3-526_01

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

June 2010
# Table of contents

TABLE OF CONTENTS......................................................................................................................... 1  
1 INTRODUCTION .................................................................................................................................. 2  
2 MOTIVATING SCENARIO .................................................................................................................... 3  
3 RELATED WORK ............................................................................................................................... 5  
4 CONTEXT AWARE ADAPTIVE SYSTEMS REQUIREMENTS ..................................................................... 5  
   4.1 CONTEXT MODEL REQUIREMENTS ............................................................................................. 6  
   4.2 SYSTEM MODEL REQUIREMENTS ............................................................................................. 8  
   4.3 SYSTEM AND ITS CONTEXT RELATIONSHIP REQUIREMENTS .................................................. 9  
5 CASE STUDIES .................................................................................................................................... 12  
6 FUTURE CHALLENGES ..................................................................................................................... 20  
   6.1 THE SYSTEM CONTEXT MODELING ......................................................................................... 20  
   6.2 THE SYSTEM MODELING .......................................................................................................... 20  
   6.3 THE SYSTEM AND ITS CONTEXT RELATIONSHIP ...................................................................... 21  
7 CONCLUSION ...................................................................................................................................... 22  
REFERENCES ........................................................................................................................................ 23
A System-Context Relationship Oriented Survey of Context-Aware Adaptive Systems

Mahmoud Hussein, Jun Han, and Alan Colman

Abstract

Self-adaptivity has been proposed to reduce the complexity associated with the management of current large scale software systems, where it provides the ability of the software system to adapt itself at runtime to cope with changes in its environment and the user needs. On the other hand, context awareness research share the same goal of self-adaptivity, but more concerned with modelling, processing, and managing the system environment information. In fact, software systems such as vehicle systems need to (1) adapt itself in response to changes in the driver needs and its environment; (2) model, process, and manage its environment information, where it is a large amount that affects the system operation. Therefore, vehicle systems need to integrate the self adaptivity and context awareness aspects, which we call context aware adaptive software system. Current research areas of self-adaptivity and context awareness are running in separated tracks, with little tries to integrate them with each other. Therefore, this research report aims to contribute to (1) a better understanding of the relationship between the context awareness and the self-adaptivity; (2) identify the requirements that are posed by integrating the two aspects; (3) analysing current research in context awareness and self-adaptivity regarding these requirements; (4) identify the research challenges in developing context-aware adaptive software systems.

Keywords: Self-adaptivity, Context awareness, Modeling, Software architectures.

1 Introduction

Recently, the demand for software systems that dynamically adapts its behavior at run-time in response to changing of the user needs and the system operational environment has been increased [1, 2]. Change is also induced by failures or the unavailability of parts of the system. Hence, it is essential for the software systems to change themselves as necessary to continue achieving their goals. Dynamic change that occurs while the system is in operation requires 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 [2]. An example of these systems is the vehicle systems, which needs to adapt itself according to the changes in driver needs, and its environment. We use a running example from the vehicle systems to help the discussion of the research report.

Research in systems that adapt to changes in order to satisfy their goals has been conducted by researchers from two communities with two different emphases: self-adaptivity [1] and context awareness [3]. The commonality between them is that they are concerned with adapting the system structure and/or behavior according to some changes. These changes can be classified as (1) Context changes: system context (environment) changes, so the system must adapt itself to cope with these changes so as to maintain its goals; (2) Requirement (goal) changes: existing requirements...
get changed and even removed, and new requirements are introduced; the system needs to adapt itself to attain these new and changed goals. Context awareness and self-adaptivity have the same goal of how to make the system performs acceptably well in case of requirements and/or environment changes, but with different emphasis in research. Research in context awareness is more concerned with how to represent, process, and manage context information, but limited on how the system adapts itself in response to changes in this context information. On the other hand, research in self-adaptivity is more about how to response to requirement and/or context changes with less attention to how context is represented, processed, and managed.

In fact, a real world adaptive system like context aware adaptive vehicle system likely need to be (1) self-adaptive, where the vehicle system needs to adapt itself as a response to changes in its environments and the driver needs; (2) context-aware, where the system need to have an explicit representation of its context, because it is a large amount of information that affect the system operation and need to be modelled, processed, and managed. In research, self-adaptivity and context awareness are often considered as a separate research areas. In practice, the line between both is rather blurred, and an engineering approach needs to consider both aspects in a holistic manner. This research report aims to contribute to (1) a better understanding of the relationship between the context awareness and the self-adaptivity; (2) identify the requirements that are posed by integrating the two aspects; (3) analysing current research in context awareness and self-adaptivity regarding these requirements; (4) identify the research challenges in developing context-aware adaptive systems.

2 Motivating scenario

Vehicle telematics is the use of computing, sensing and telecommunication to provide services to vehicle driver. It is used to improve road safety, vehicle efficiency, and ease driver stress [41]. Automotive IT researchers are exploring new ways in which intelligent vehicles (vehicle with context aware telematics device) can (1) communicate with each other to improve spatial awareness and traffic management; (2) communicate with the road infrastructure system to provide the driver with accurate up-to-the-minute information about the road and the traffic conditions; (3) collect some information about the environment using in-vehicle sensors to adjust the vehicle in its lane and to detect obstacles; (4) communicate with service providers to use the available services like services for providing nearest petroleum station, restaurant, and hotel that match user preferences. Vehicle telematics systems can be classified into four groups based on their use and the systems’ communication range [14] as in-vehicle, vehicle-to-vehicle, vehicle-to-infrastructure, and vehicle-to-service provider telematics systems, as in Figure 1.

![Figure 1: Overall View of Automotive Telematics Systems](image)

In-vehicle (INV) telematics support seamless interaction between a user’s portable devices and In-vehicle telematics. This allows the driver and other occupants to be entertained or to carry on their
work. For instance, a passenger can use his laptop together with the vehicle wireless internet broadband to carry out his work. Also, the vehicle can use its internal sensors to collect information about its environment, where it can use (1) the rain sensor to detect rain status and adjust its speed accordingly to that; (2) the vehicle internal camera to detect the driver status. **Vehicle-to-vehicle (V2V)** telematics support interaction amongst vehicles, where they can talk to each other to share safety warnings, collaboratively detect hazards, and avoid collisions. **Vehicle-to-infrastructure (V2I)** supports interactions between the vehicles and the roadside units for exchanging the traffic information, and in the event of an accident, a vehicle could automatically obtain information about nearby vehicles as a means to identify potential witnesses. **Vehicle-to-service provider (V2SP)** telematics support interaction between vehicles and the service providers through networks like 3G. The driver can obtain information and access to various value-added services, can be or not directly related to travelling such as news and infotainment.

The following scenario can happen in such a vehicle system, and shows the need for integrating the context awareness and self-adaptivity aspects. The scenario is about James who is driving his vehicle home. His context aware vehicle is Bluetooth-, DSRC-, 3G-, and GPS-enabled. It provides several applications like (1) **Vehicle-Navigation-Assistant (VNA):** assists James with route planning, detects/avoid congestions, and can communicate with other vehicle as leader or follower to form a cooperative convoy; (2) **Vehicle-Services-Discovery (VSD):** provides access to external services like radio, internet, commercial services that match the driver’s preferences such as service that provide the locations of nearest petrol stations, hotels, restaurants, and pharmacies.

**Scene 1:** The scenario starts when James entering his vehicle. The vehicle detects James, says “Welcome James;” and activates his profile. The vehicle starts to adjust air-condition temperature, 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 wants to plan his way home that requires a navigation service. Therefore, the vehicle system (1) loads the VSD application to search for and communicate with the best available services that provide navigation and pharmacies locations, while considering his preferences (e.g. medium navigation service quality, lowest service price, and etc.); (2) communicates with the nearest roadside unit to get the traffic information; (3) loads the VNA application which based on current context (James preference, vehicle location, pharmacy location, and traffic information), it shows the available routes for James to choose the suitable one. Finally, before starting the journey the VSD is unloaded to reduce its overhead and improve the system performance.

**Scene 3:** While James was driving, a rain is detected by some dedicated sensors. He is informed of the speed limit, and he changes the vehicle speed according to that. When the rain becomes heavy the vehicle communications with the road side unit and the service providers are disabled. He becomes unable to (1) reach the navigation service; (2) obtain the traffic information. To use a navigation service and obtain up-to-minute traffic information, his vehicle (i.e. as a follower) communicates with one of the nearby vehicles (i.e. as a leader) that has the same route and still able to communicate with a navigation service provider and a road side unit. In this convoy, the two vehicles need to (1) exchange the distance between them every 10 seconds for making this cooperation safe; (2) coordinate their speeds to keep the communication link between them as much as possible. Then, James continues his journey home.

The above scenario shows that the context aware adaptive vehicle system should integrate the context awareness and self-adaptivity aspects where it needs to:
model, process, and manage its context information, where (a) the system has a large amount of information about the driver, the vehicle, and the vehicle environment (e.g. the nearby vehicles and the services providers) that affect the system operation; (b) the vehicle sensors return a raw data about the environment that need to be processed to infer high level information; (c) the environment information needs to be managed between multiple applications with different needs. (d) the context providers can come and go (e.g. the road side units), and the switching between the context providers should be performed at runtime; (e) the system need to interact with other systems (e.g. nearby vehicles), therefore it should model its interactions with them to interact effectively; (g) the system needs to be aware of the available resources (e.g. service providers) to take the correct actions.

adapt itself in response to the changes in the driver needs and its environment, where the vehicle systems’ structure and behaviour adapt itself in response to (a) context changes where, (1) if the driver specified in his diary to get the medication, then the system structure is changed by adding the VSD to search for the service that provides the nearest pharmacy location; (2) the vehicle speed is changed in response to a rain detected; (b) driver needs changes where, (1) if the driver needs to plan his journey, then the VNA is loaded; (2) during the journey the VSD is unloaded, if it is not needed by the driver to improve the system performance; (3) in response to a request from the driver, the time between each vehicles speed exchange in the cooperative convoy can be increased or decreased.

3 Related Work

Several surveys from different viewpoints have been carried out such as surveys on context modelling techniques, system modelling techniques, context aware systems and self-adaptive systems. From the modelling perspective Bettini et al. [4], Bolchini et al. [5], and Strang et al. [6] surveyed and evaluated the proposed context modelling techniques. In the other hand, Medvidovic et al. [7], and Clements [8] have specified the required features for the software architectures as a system modelling technique and then analysed the proposed architectural description languages. These surveys are related to one aspect of the relationship between the system and its context, which they are concerned with the context or the system modelling techniques.

From the systems perspective, Matthias et al [3] 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, where this type of research is more concerned with how to represent, 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. [9], Nami et al. [10], and Bradbury et al. [11] have surveyed the proposed self-adaptive software systems, but they don’t consider the requirements for integrating the context awareness with self-adaptivity. In general, these surveys have not systematically considered the requirements for the integrated modelling of the system and its context for developing the context aware adaptive software systems.

4 Context Aware Adaptive Systems Requirements

The above scenario shows that there is a need for integrating the context awareness and self-adaptivity aspects. Therefore, in designing this type of systems, there is a need to have an integrated modeling for the context, the system and their relationship. In this section we discuss the requirements that posed by integrating the two aspects.
4.1 Context Model Requirements

The term context-aware appeared in [12] for the first time, where the application is aware of context like locations and identities of nearby people, and objects. Later, one of the most accurate definitions is given in [13] 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 (i.e. the application), 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 system is going to operate in relation with the other system (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.” Several authors propose different requirements for context modeling from the context perspective [4-6, 14], but here we concentrate on the requirements that are posed by the context relationship with the system as follow:

4.1.1 Context modeling aspects:

*Physical context:* which represents facts about the system environment that affects the system operation such as (1) problem (strategical) context, which is the information that initiates the system adaptation actions (e.g. in case of low fuel level, the system structure is changed by communicating with a petrol station locator service to find the nearest petrol station location); (2) operational context, which can be used (a) to initiate a system function execution (e.g. in the cooperative convoy the leader car speed change initiates an interaction with the follower car to exchange the vehicle speed); (b) as parameter that is needed by the system to operate (e.g. to find the nearest pharmacy the vehicle location is needed). Therefore, there is a need to represent the problem and the operational context. In the other hand, the solution context is more related to the system modeling, which represents the available resources that can be used by the system. Therefore, it will further discussed as system modeling requirements.

*Situational context:* Environment sensors returns raw context data (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 application. This information (i.e. situational context) is the aggregation of multiple context data to conclude high level information based on one or more context entities low level information with respect to an entity that is interested in this situation (e.g. the driver is interested in harsh weather which can be inferred from the temperature value, the rain status and the wind speed). Therefore, it is important that the context model should be able to represent high level information.

*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 model should enable the representation of (1) complex relationships (i.e. the interaction-oriented context) where a set of rules, obligations and understandings that influence an individual’s action with respect to a group in a particular situation, for making the system able to cooperate with other interacting systems (e.g. the service providers). Furthermore, it can be between the computational context entities themselves that is relevant to the system operation (e.g. the interactions between the driver and the vehicle where he can be put the seatbelt on); (2) simple relationships, where not only there is
a need to represent the interactions between the system and its computational context entities, but also the there is various relationships between the physical non-computational context entities themselves that affects the systems operation that need to be modeled such as (a) nearby relationship between a person and a device; (b) engaged in relationship between the person and an activity; (c) dependence 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); (d) located at relationship between a person and a room.

Subjectivity: In the interaction oriented context, actors may have their own perception of the group and then the context can be modeled subject to one of these actors (e.g. one of the vehicles in the cooperative convoy) or from the domain where they interacting in (e.g. the traffic management system). In the other hand, the physical context can be modeled subject to the user, the application, or the organization the user works in if any. Therefore, context modeling should enable the context representation with respect to (1) one of the interacting entities; (2) the domain where context entities are interacting in.

Information quality: Context information can be of varying quality, depending on its source. 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, where information correctness is an important factor.

4.1.2 Context management:

Runtime representation: Not only modeling the semantics and the relations between context information at a conceptual level is very important (i.e. ontology), but also the runtime representation of the context data is needed to ease the developer tasks by automatically generating the required code that is corresponding to each context entity (i.e. model driven development). Therefore, the context entities should be represented at runtime as first class entities and not be embedded into the application code to reduce the application complexity.

Context history: 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. Furthermore, context information can be stored to be used by the system (e.g. driver preferences and diary). Therefore, there is a need to store the context information for future use or context values predication.

Runtime evolution: As the system and its context need to evolve, so the context model should have the ability to evolve to cope with unexpected environment change. Therefore, context models should be flexible and extensible, where new context elements shall be allowed to enter the system environment.

Reasoning: While the system needs high level context information (e.g. is the vehicle is nearby) and the sensors return low level context data (e.g. the coordinate of this vehicle). Therefore, the model should support the reasoning to obtain (1) the situation the system is in (i.e. situational context); (2) the quality of the provide context to help in selecting the context provider (i.e. information quality); (3) the relations between the non-computational context entities (i.e. simple relationships).

Context sharing: The above scenario shows that (1) the system can have multiple applications that use the same context data such as the vehicle service discovery and the vehicle navigation assistant; (2) context can be shared between multiple interacting systems such as between the two vehicles in the cooperative convoy; (3) the nodes that carrying the context information are distributed such as road side units, nearby vehicles, and etc. Therefore, the context modeling language should have a common vocabulary to enable this sharing and hide the context heterogeneity.
4.2 System Model Requirements

The system can be modeled using design methods [15], module interconnection languages (MIL) [16], software architectures [17], or design patterns [18]. Due to the increased complexity of the software systems, the trend is to use the software architecture which aims to specify the system structure at a sufficiently abstract level to deal with large and complex systems. The Architectural Description Language (ADL) are used to describe the software architecture and to support the reasoning together with the software construction, which it can support multiple views [19]. These views can be presented as annotation to a shard architecture model, e.g. for the behavior modeling, each component can be elaborated with its behavior and the overall behavior can obtained using the composition of the multiple components behaviors. Therefore, the same architecture (i.e. the system structure that represents what the system should do) can be used to support multiple views such as (1) the behavior view that represents the system functionality under the execution (i.e. system dynamics), and (2) the performance, reliability, and security views for representing the system non-functional requirements (i.e. how well the system functionality should be performed) that need to be maintained during the software execution. Several authors discussed the requirements for the traditional [7, 8] and adaptive [20-22] system modeling, but our concern is the adaptive system modeling aspects in relation to its context. Therefore, the following are the requirements that need to be considered form the system modeling perspective:

4.2.1 System modeling aspects:

Structure and behavior models: In response to context change the system adapt its structure (i.e. re-configuration) and/or its parameters (i.e. regulation, where the system behavior is adapted) to cope with this context change while preserving its high level goals. Therefore, the system should be modeled from the structure and the behavior perspectives, while considering the system evolution in mind (i.e. adaptable system structure and behavior). Furthermore, the system functional requirement model needs to be maintained at runtime to assure the system adaptations don’t violate its goals.

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 (i.e. security, performance, and reliability) should be preserved before, during, and after performing the adaptation. Therefore, the non-functional requirements models need to be maintained at runtime.

Solution context: During the runtime the system structure adaption is performed by removing, adding, or replacing a system component(s). To perform the addition and replacement adaptation operations, the system need be aware of currently available components (i.e. the system inactive components) that can be used. Furthermore, to change the system parameters (e.g. the number of used units of the CPU, and the memory); there is a need to know the permitted parameters changes (e.g. the CPU usage can only be increased by 10%). Therefore, a model for the system available resources should be maintained.
4.2.2 System models management:

**Reasoning:** to build a correct context aware adaptive system, its models should have a formal representation that enables validating the system structure or 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 enable the initiation of the adaptation actions in case of consistency violation.

**Runtime adaptation:** The system will have a long life time. During this time the system need to adapt in response to context change while it is in operation. Therefore, the system structure and its behavior need to have the ability to be changed at runtime for making the system able to handle these new situations and survive for a long time. Furthermore, changing the system structure and its behavior models at runtime means that any change in one view should be reflected to the other view to keep them consistent.

**System generation:** The usefulness of system models is directly related to the kinds of the tools it provides to ease the developer tasks such as system design and executable system generation. Therefore, a tool is needed to automatically generate the system prototype from the designed model (i.e. the system structure and behavior). Furthermore, the functional and non-functional requirements models should have their corresponding code generated automatically to be maintained at runtime.

4.3 System and its Context Relationship Requirements

The relationship between the system and its context is shown in Figure 2. The context poses three requirements into the system: (1) **monitoring,** where the system should specify the environment elements (i.e. the problem, operational, and the solution context) that need to be monitored, where their states are reflected into the context model. In the same manner the running system state is reflected into the system model; (2) **adaptations** (i.e. context switching), where in case of context change the system adaptation component decides the required adaptation actions in response to this change based on the currently available system resources (i.e. the solution context), the running system state (i.e. the system model) while preserving the system functional and non-functional requirements (i.e. goals, performance, and etc. models); (3) **Acting,** where system adaptation leads not only to system changes but also its environment (i.e. the solution context) via the effectors, where the system can use the current available resources and then it moves from the solution context to the system or vice versa when the system component is removed or replaced where it moves from the system to the solution context.

![Figure 2: The system and its context relationship](image-url)
The adaptive system can be designed as a set of states (i.e. non-adaptive programs) and the transitions (i.e. a set of adaptation actions) between them shows the system adaptive behavior [23]. Not only the context changes need to be handled, but also the system requirement changes. Therefore, Figure 3 shows all the combinations of context and requirements (i.e. functional and non-functional) changes. When the system is in state S1, where it satisfies a set of requirements (R1) and current environment state (context1), it can move to another state base on: (1) requirements change (R2), where the system moves to the state S2; (2) context change (context2), where the system moves to the state S3; (3) context and requirements change (context2, and R2), where the system moves to the state S4 to cope with these changes. Here we more concerned with the context change, where we consider the requirements are not changeable and need to be preserved when the system is adapted.

Figure 3: Context, requirements, and system change possibilities

There are two types of context that need a response from the system (1) operational context, where it only need to be monitored for the usage by the system; (2) problem context, where it needs not only to be monitored but also the system need to adapt (i.e. deciding the required adaptation actions and acting them). To capture the relationship between the system and its context as the system response (i.e. acting the change on the solution context) to its context change (i.e. monitoring the problem context) using decision making mechanism (i.e. adaptation), the following requirements should be considered:

4.3.1 Monitoring

Monitoring technique: To know the current context state (i.e. maintaining up-to-minute context model), a monitoring technique is needed. There are three methods for monitoring: (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.

Optimal monitoring (i.e. less overhead and cost): The space of monitorable entities can be huge, but in any time the system doesn’t need to monitor all the context entity where there is a number of applications with different context requirements or each application has its changeable context requirements. Therefore the system needs to have an adaptive context awareness to minimize the monitoring overhead and cost.

Efficient context change detection: After monitoring the system context and reflect it to the context model, there is a need to have a method that is used to evaluate the context model to infer context change that needs a response from the system. Therefore, an efficient evaluator is needed to detect this change in small time.
4.3.2 Adaptation

**Adaptation mechanism:** It is represented as policy that guides the determination of the required adaptation actions in response to context change (i.e. moving from one state to another). Kephart et al. defined the policy as “is any type of formal behavioral guide” [24]. They listed a three types of polices that can be used: (1) *action policy*, which takes the form of “IF (condition), THEN (actions)”, these rules are pre-computed which the designer should specify; (2) *goal policy*, which rather than specify what to do in current state like action policies, a goal state is defined and then an algorithm is used to infer the required adaptations actions; (3) *utility policy*, which generalize the goal polices and instead of classifying the states as desired next state or not (binary classification, i.e. 0 or 1), a real value is given to each possible state.

**Support large number of adaptations:** Software systems moves from small size to a large scale system that needs a good mechanism to handle larger number of adaptations, where there are a large number of environment changes with their corresponding system changes. Therefore the method used to capture this relationship should handle this.

**Adaptation mechanism efficiency:** Adaptation decision must be performed in specific time no more than it, specifically in safety/mission critical systems, where late adaptation actions will not have any value. Furthermore, the decision making that takes long time affects the system functionality, where the system spends its resources in doing adaptation without taking care of doing its functionality. Therefore, the system needs to decide the required adaptation actions to satisfy the new context before another change occurs.

**Automated decision making:** Adaptive system is deployed in unexpected environment, where the communication with the system developers is infrequent. Therefore, the decision must be performed automatically without (with little) human guidance where the system drive automatically the new adaptation actions to cope with unexpected environment changes. Furthermore generating on-the-fly adaptation actions can (can’t) (1) achieve what it is intended to do, so it need to be stored for using (avoid using) in the future; (2) be a valid adaptation actions, so it needs to be verified before applying it.

**Adaptive behavior realization:** In large scale systems writing adaptation polices is error prone and writing the code corresponding to them is complex. Therefore, the adaptation policies should be easy to write and can be transformed to a code that represents the system adaptive behavior for complementing the system generation process.

4.3.3. Acting

**Acting method:** After the adaptation component decides the required adaptation actions which may contains action that change the solution context by adding component(s) that is removed from the system or vice versa, these adaptation actions need to be applied into the running system. There are two techniques that can be used (1) *model based*, where the adaptation actions are applied to the system/solution context models and then these changes are reflected to the running system and its solution context; (2) *non-model based (direct)*, where the adaptation actions are applied directly.

**Safe change management:** System need to be in operation as much as possible to reduce its downtime, so the adaptation needs to be performed at runtime while preserve the consistency of the system state. Therefore, there is need for a mechanism for safely applying the adaptation actions to keep the system in a consistent state before, during, and after adaptation to increase the trust of users in the system they use.

**Concurrent adaptation:** Due to high number of context changes in small time as the property of the vehicle systems, the system may have multiple adaptation actions that need to be performed concurrently. In case of multiple adaptations pending, this pose the following questions that need to
be answered: which one to chooses, can a running adaptation be stopped to run a high priority one, can the system rollback itself to a known consistent state when the current adaptation action is stopped and another one will be executed. As such, a good scheduling algorithm is required to coordinate between multiple adaptations, while leaving the system in consistent state after performing the adaptation.

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. Here we will briefly overview several such research. The section presents a collection of notable highlights, where it lists the most recent techniques that have been introduced.

The Rainbow framework [25] 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) select the required adaptation actions; (4) effect the needed changes to the running system. They use Acme [26] to model the system structure and its environment. The Acme language has been proposed for describing the system architecture and to enable architectures that designed with different architectural description language to be exchanged. This makes it too general, and it should be extended to enable the representation of the specific system elements. The system context is represented as a set of resources (i.e. the solution context) that support the adaptation process and the dependences between them can be represented as model constraints. Furthermore, modeling the environment as the Acme architectural model enables runtime representation and evolution and the context sharing between multiple interacting systems. In the other hand, it is not suitable to (1) model the interaction oriented context; (2) the simple relationships such as nearby and engaged in between the context entities themselves; (3) deal with incorrect information where they assume the system probes provide correct information about the system and its environment. They model the system structure for enabling its construction and it is maintained at runtime for performing models analysis to initiate the adaptation process. The Acme itself can’t be analyzed but it can be mapped to other architectural description language (i.e. WRIGHT) to enable verification. They use probes to monitor the system context which they can be deployed as needed to achieve optimal monitoring, but their model evaluator is based on models constraints violations which it is problematic in large scale systems. They use language called Stitch to capture the relationship between the system and its environment changes (i.e. the adaptation policy), but modeling the adaptive behavior manually using the Stitch language is a problem in large systems where there are many variations needs to be captured. When adaptation action is applied to the system, its models are evaluated to check is the intended adaptation goal is achieved or not, but they don’t show how to safely apply the needed change. Furthermore, they extend the framework through an approach in which adaptations can be preempted to allow other time-critical adaptations to be scheduled [27].

MUSIC project [28] is a component based framework that is used to optimize the system overall utility in response to environments changes. They have 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 change. They propose an environment model that is based on ontology for modeling the context information. This model facilitates the model driven development of context aware 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 problem context), with less attention to modeling the system resources that support the adaptation
process in choosing the correct adaptation actions. Subjectivity is supported where the context can be modeled from the application perspective or for the user perspective. They support the context awareness model to be (1) changed at runtime, but also the context models need to be changed; (2) used to achieve optimal monitoring, where only the required context elements are monitored. Their approach is based on ontology which support the reasoning process, but ontology based techniques have a bad performance in reasoning complex situations. Their system model is based on a system component meta-model proposed by them which represents the system structure, and it can be used to design a specific system model that conforms to it, and then the system code can be generated. In the other hand, they don’t have a technique for verifying the system model regarding its constraint. They capture the relationship between the system and its context by the context values that are defined in the utility function of each component. Different component implementation variants are evaluated to choose the one with highest utility. The developer should design the utility functions that mean it is a problem in large system where there is a complex system behavior and utility functions.

Heaven et al. have developed an approach for building self-adaptive systems, where the system adapts its structure in response to environment change to achieve high level goals. They used Labeled Transition System (LTS) to model the system domain [29]. They represent the environment at high level of abstraction to enable top-level planning (i.e. to generate the adaptation plans). This model captures the states the system and its environment can be in, and the environment changes are as the actions that move the system from one state to another. In their technique, the context information is embedded into the system domain model which it is not suitable to model (1) the interactions oriented context; (2) the solution context where only the problem context is captured. The system structure is modeled based on the Darwin component model which represents its software and hardware components, and there is a tool support for generating the system prototype from this model. The relationship between the system and its context is embedded into the domain model, where context change moves the system from one state to another. The LTS is based on the system and its environment variables, and then in large system where there are a large number of variables that describes the system and its environment with different variations in their values, it is difficult for the designer to build a complete and correct LTS. If the designer have the ability to build the LTS, it will be difficult to validate because of the state explosion problem. Furthermore, they have a technique to safely apply the adaptation action which changes are applied to a model and then reflect to the running system (i.e. model based) while preserving the system state consistency.

Morin et al. [30] propose a technique to handle the exponential growth of the number of configurations that are derived from the system variability. 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 that used in the adaptation rules (i.e. the problem context). They didn’t have a separate context model, where the context information is modeled within the system model, and it is looks like the key-values models (i.e. attribute-values pairs) which fail to capture the solution and the interaction oriented context, and the context data quality. Furthermore, they don’t have a runtime representation of the context model at runtime, and then they can’t modify it at runtime to cope with unexpected environment change. Their structure model is based on meta-model proposed by them which can be represented at runtime and it can be validated regarding the system invariants. They capture the relationship between the system and it context using the adaptation rules where in context change these rules is evaluate to (1) identify the system structure change; (2) guide the selection of the system feature using a set of rules and then based on select features the system structure is changed.
An approach was introduced by Cheng et al. for creating formal models for the self-adaptive system behavior [23]. In their approach the system adaptive behavior is separated from the non-adaptive behavior. This separation makes the system models easier to specify and can be verified. They use Petri-Nets to model the system adaptive behavior, where they use context change as guidance for the transition between the system states, so they implicitly consider the context as a part of the system adaptive behavior model. This technique in modeling the system environment is not rich enough to capture the interaction oriented and the solution context, and the context information quality. Furthermore, there is no explicit context model maintained at runtime, therefore the context

---

**Table 1: Comparison between current research according to our requirements**

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Current research</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Modelling aspects</strong></td>
<td></td>
</tr>
<tr>
<td>Physical context</td>
<td>O O O O P/O O O O</td>
</tr>
<tr>
<td>Situational context</td>
<td>++ ++ ++ ++ ++ ++</td>
</tr>
<tr>
<td>Relationships</td>
<td>+ + + + + + + + +</td>
</tr>
<tr>
<td>Subjectivity</td>
<td>+ + + + + + + + +</td>
</tr>
<tr>
<td>Information quality</td>
<td>++ ++ ++ ++ ++ ++</td>
</tr>
<tr>
<td><strong>Management</strong></td>
<td></td>
</tr>
<tr>
<td>Runtime representation</td>
<td>- - ++ ++ ++ ++ ++</td>
</tr>
<tr>
<td>Context history</td>
<td>++ ++ ++ ++ ++ ++</td>
</tr>
<tr>
<td>Runtime evolution</td>
<td>++ ++ ++ ++ ++ ++</td>
</tr>
<tr>
<td>Reasoning</td>
<td>++ ++ ++ ++ ++ ++</td>
</tr>
<tr>
<td>Context sharing</td>
<td>++ ++ ++ ++ ++ ++</td>
</tr>
<tr>
<td><strong>Modelling aspects</strong></td>
<td></td>
</tr>
<tr>
<td>Structure and behaviour</td>
<td>S S S S S S S S S</td>
</tr>
<tr>
<td>Quality models</td>
<td>- - - - - - - - +</td>
</tr>
<tr>
<td>Solution context</td>
<td>- - - - - - - - +</td>
</tr>
<tr>
<td>Reasoning</td>
<td>- - - - - - - - +</td>
</tr>
<tr>
<td>Runtime adaptation</td>
<td>- - - - - - - - +</td>
</tr>
<tr>
<td>System generation</td>
<td>- - - - - - - - +</td>
</tr>
<tr>
<td><strong>Monitor</strong></td>
<td></td>
</tr>
<tr>
<td>Monitoring technique</td>
<td>E/C C C E/C C C C</td>
</tr>
<tr>
<td>Optimal monitoring</td>
<td>+ + ++ ++ + + + +</td>
</tr>
<tr>
<td>Efficient evaluator</td>
<td>+ + ++ ++ + + + +</td>
</tr>
<tr>
<td><strong>Adaptation</strong></td>
<td></td>
</tr>
<tr>
<td>Adaptation mechanism</td>
<td>- - - - - G G A A A A A U U A U A U A A A A A U U</td>
</tr>
<tr>
<td>Large number of adaptations</td>
<td>- - - - - + + ++ ++ + + + + + + + + + + + + + + +</td>
</tr>
<tr>
<td>Adaptation efficiency</td>
<td>- - - - - - - - - - - - - - - - - - - - - - - - - - - - -</td>
</tr>
<tr>
<td>Automated decision making</td>
<td>- - - - - ++ ++ ++ ++ ++ ++ ++ ++ ++ ++ ++ ++ ++ ++ ++</td>
</tr>
<tr>
<td>Adaptive behaviour realization</td>
<td>- - - - - ++ ++ ++ ++ ++ ++ ++ ++ ++ ++ ++ ++ ++ ++ ++</td>
</tr>
<tr>
<td><strong>Acting</strong></td>
<td></td>
</tr>
<tr>
<td>Acting method</td>
<td>- - - - - M M M M M M M M M M M M M M M M M M M</td>
</tr>
<tr>
<td>Safe change management</td>
<td>- - - - - ++ ++ ++ ++ ++ ++ ++ ++ ++ ++ ++ ++ ++ ++ ++</td>
</tr>
<tr>
<td>Concurrent adaptations</td>
<td>- - - - - - - - - - - - - - - - - - - - - - - - - - - - -</td>
</tr>
</tbody>
</table>

(-) Unsupported  (+) Partially supported  (+++) Fully supported  (P) Problem context  (O) Operational context  (S) Structure model  (B) Behavior model  (C) Continuous monitoring  (E) Event monitoring  (A) Action policy  (G) Goal policy  (U) Utility policy  (M) Model based  (D) Directed
management operations can’t be performed. They concentrate on the system behavior change, where they model the system behavior only. Regarding the system and its context relationship, they defined how the system can safely moves from one state to another in response to context change using the Petri Nets transition operation. Modeling the adaptive behavior as state based model has a difficulty in specifying large systems, because the number of system states is growing exponentially.

Folch et al. proposed a technique to use architectural models for runtime adaptability [43]. Their architecture model consisted of the system basic component and for each component a set of alternative implementations is defined. For each component a utility function is defined to evaluate the different component implementations 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 don’t specify what the type of their context model is. It seems that they use a simple model for the needed attributes and their values. Therefore, this model is not rich enough to satisfy the context modeling aspects and management requirements. They capture the relationship between the system architecture change 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 that means it is a problem in large system where there are complex utility functions.

Henricksen et al. [33] present a software framework that addresses the challenges of building context aware systems. 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 not capture the interaction oriented context and can’t be used to handle unexpected context change where the context model is not adaptable. As the SOCAM project [31], 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 [42]. 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 unexpected 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.

Where the space of observable entities of context aware systems is huge and each application has specific monitoring requirements, therefore CA3M [32] which follows the model-driven engineering approach is used to help the designers to specify context-awareness concerns into models that conform to their context-awareness meta-model. Their context awareness model has a runtime representation and it 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 pay attention to how the system going to react in case of context change,
but this research is in its initial stage where they intended to extended the work to include the system structure and behavior adaptation as mentioned by the authors.

Oreizy et al. proposed an architecture-based approach for runtime software evolution [35]. 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 evolution management, as the process of applying the changes to the running system. They observe the system to know the 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 can’t be represent; (2) there is no runtime representation of the context model to enable coping with unexpected environment changes. Their approach is architecture-based where an explicit architectural model is maintained at runtime and changes 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. 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.

Sheng et al. proposed an approach for developing context-aware web services. Their development environment (ContextServ) relies on a UML meta-model called ContextUML [34]. This meta-model is based on four concepts (1) context element, which represent the web service environment and it can be simple element (i.e. one context element) or composite element (i.e. the aggregation of multiple simple elements). Modelling the context as UML elements doesn’t provide the ability to capture the interaction oriented context and the subjectivity aspects. Furthermore, they have a runtime representation of the context model, but it is not adaptable to enable the handling of unexpected context change; (2) context sources element, which specifies what are the context entities to be monitored for (b) specifying the monitored context entities for each service separately; (c) the switching form a context provider to another based on the provided context quality at the runtime; (3) service modeling element, which specifies the service operations, inputs, and outputs 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 for enabling structure change. Furthermore, they don’t maintain the system behaviour model; (4) context awareness element, which specifies the service and its context relationship and it is classified into two types: (a) context binding, where the service needs the context attribute value as parameter; (b) context triggering, where in response to context change an action is taken to adjust the service operations.

Medvidovic et al. proposed an extension to the C2 architectural style [44]. 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; therefore they extend it by adding the side connector to enable components interactions. These interactions are initiated based on providing or requesting a service via event-based communication. Furthermore, to solve the distribution problem, the 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 it is a representation of the system. They assume that the correct system state is reflected into the model and then they don’t address how to assure the correctness of this information. This meta-level is used
know the system status or for performing the required adaptation action by apply the required changes to this model and then reflected them to the running system. 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 don’t have an explicit environment model at runtime which limit the system ability to cope with unexpected 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 as adding, removing, and replacing system components or connectors. Furthermore, they don't maintain a behavior model, where they are not concerned with adapting the system behavior.

The CAWAR framework is service based approach [45]. This framework is based on a context adaptation meta-model. This meta-model has four types of service: (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. In the other hand, this model is not suitable to model the interaction oriented context. They use another service which called model activator that (1) implements the adaptation decision making mechanism as an action policy, and (2) supports the inference about the system and its context models to initiate the adaptation process. Their framework can be used to adapt the core system and the adaptation logic to support unexpected changes, which they call the total reconfigure-ability. The operation of coping with unexpected environment changes is performed manually where in case of new context element; a new system configuration is designed and the adaptation logic is changed to link the context change with this new configuration. Therefore, in large systems where there are a large number of adaptations it is not possible to model all the possible variations.

The specific technical details of context aware middleware limit the ability to develop an application that can be run over different middleware. To solve this problem, Ayed et al. [40] extended the UML class diagram to incorporate the definition of context information that is relevant to the application. They add the following stereotypes (1) context, which used for defining the simple context elements such as the location and the bandwidth, but it is not suitable to model the interaction oriented context and the context entities simple relationships; (2) collection process, that collects the required context information and it can be event based or periodic monitoring (i.e. continuous); (3) context quality, to indicated 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 application. Another extension is performed for both class and sequence diagrams to incorporate the system adaptive structure and behavior, where they add (1) variable structure, which defines several variants to the same function or an optional function that can be added or removed; (2) variable architecture, which some components are defined as optional that can be added when needed; (3) variable sequence, which specifies different sequences in the sequence diagram (i.e. sequence variant) for each possible system behavior. To build a correct adaptive system and reason about it using this UML notation, a formal representation is needed which they don’t provide it. 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 is intertwined with the system model (2) limits the ability to cope with unexpected environment changes where the relationship should specified at design time; (3) has the difficulty in modeling large system where there is large number of variations that needs to be considered.
To cope with unexpected environment changes, the adaptation logic should have the ability to be changed at runtime. Therefore, Andrade et al. proposed an approach to do this by separating the adaptation logic from the core system artefacts, and then these autonomic artefacts can be changed at runtime [36]. 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. It is designed as component based, where each monitor is a component that can be added or remove. 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 policies 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 by the same manner like the environment module which it is a component based to enable dynamic change; (3) redeployment, which is responsible for applying the required adaptation actions to the running system. It contains two methods: handle adaptation to perform pre-deployment operations for ensuring safe adaptation actions and 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.

The realization of self-adaptive system have many challenges that posed by the system complexity and its adaptive behaviour, therefore the StarMX has been proposed to ease the task of realizing these type of system based on java [37]. Their framework is based on (1) a rule engine for the reasoning and the decision making; (2) JMX technology for the sensing and effecting purposes. The StarMX architecture is consisted of two main elements: (1) execution engine, which contains the self-adaptive operations (i.e. monitoring, adaptation, and effecting) as processes and each process is as an execution chain that define how to perform the required operations. Each process (e.g. effecting) is linked with specific system elements (e.g. a specific effectors that created by the system developer) that provide the needed function via an anchor object; (2) set of services, which are used to enable the execution of the of processes such as (a) lookup service to access the anchor objects; (b) activation mechanism for triggering the execution chain 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 a process for engineering a system with the self-adaptive properties with the StarMX as follow (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) defining the system adaptation polices; (4) configure the StarMX by linking what was defined in the previous steps with the running 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.

To assure that the users get the best quality of service, the provide services need to aware of the environment that is running in (i.e. context-aware) and adapt itself in response to context change (i.e. adaptive). The PLASTIC project [38] aims to support the adaptation of its service based on context or QoS change. 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 support the model of the service quality attributes like performance and reliability, and the service behaviour model. The environment elements that they are concerned with are the
available resources such as bandwidth, CPU time, and memory. Therefore, they monitor these parameters directly without having a context model. Considering the system environment internally limits the ability to (1) represent the interaction-oriented context, context entities simple relationships, context quality; (2) have a context model runtime representation to enable context evolution, reasoning and sharing. They capture the relationship between the system (i.e. service) adaptation and its context or quality of service changes by adding annotations to the system variants, where each variant has his resource demand and the offered quality, and then based on the user context and the request quality of service, the suitable variant is chosen (i.e. utility policies). 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, safe change management, runtime adaptation, concurrent adaptations scheduling and optimal monitoring are not supported.

André et al. proposed a generic framework for the adaptation of web services [39]. They use probes (i.e. hardware 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 system monitors to infer high level information from these low level data. The collect context information are passed directly to the adaptation manager without having an explicit context model at runtime. Therefore, their technique is not able to (1) cope with unexpected environment changes; (2) represent the interaction oriented context and information quality; (3) share the context information between multiple systems. 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 relation between the context change and the system adaptation, they use (1) action policies, where adaptation actions are specified beforehand to have a fast system reaction; (2) utility policies, to handle the unexpected dynamic change, where the specific adaptation actions are not specified until the runtime. Their decision making mechanism can be distributed between multiple cooperative sub-systems, therefore they can handle concurrent adaptations.

Applications are deployed in environment that has dynamic resources (i.e. limit resources or resources with different reliability), therefore the applications need to detect the changes in the resources and adapt itself according to that. The CASA is a contract based adaptive software architecture framework that is proposed to achieve this requirement [46]. The system (i.e. the application) 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 have a runtime change to the adaptation logic, 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, context entities relationships and the context quality indication can’t be represented where they don’t have a context model. In the other hand, they have a resource manager that maintains the system resource status and it has the ability to cope with resources evolution at the runtime. Their method for modeling the system doesn’t (1) have formal representation to enable the reasoning; (2) support the model driven development where the system code can be generated; (3) support the model the system behavior.
6 Future challenges

In section 5, we analyzed current research in context awareness and self-adaptivity to know how they are satisfying our needed requirements that are listed in section 4. In this section, we use our requirements (i.e. context modeling, system modeling, and their relationships) to structure the presentation of the future research challenges. Some research started to address these challenges, so we refer to what they have done to ground our discussion.

6.1 The System Context Modeling

Modelling aspects: The system need to interact with its context entities (e.g. the nearby vehicle in the cooperative convoy or the service provider that provides the navigation service). These interactions affect the system operation. Therefore, modelling these interactions is important to ensure that the system is operating correctly. Also, 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. Furthermore, most of currently introduced context modelling techniques consider the problem context with less attention to the solution context. However, the system needs to be aware of the available resources to take the correct adaptation actions. Therefore, the context model needs to have the ability to represent the solution context.

Context management: Current research in context aware adaptive systems captures the system environment using two ways: (1) implicitly, where environment information is considered as a part of the adaptation technique (i.e. rules, strategies or finite automate) [23, 29, 30] without having an explicit model for representing the system context. In these techniques the system context is monitored and the environment values are passed to the adaptation manager to take the required adaptation action. This technique is not efficient in complex environment where the space of monitorable entities is huge. Furthermore, it is unable to handle unexpected environment changes while the system is in operation; (2) explicitly, where an environment model is maintained and is evaluate to initiate the required adaptation actions or queried to support the adaptation process [25, 28, 31-33]. Maintaining an explicit context model (1) supports the adaptation process by capturing the solution context; (2) initiates adaptation actions based on context change efficiently; (3) reduces the complexity that posed by intertwined the context information with the adaptation manager; (4) handles unexpected environment changes at the system runtime; (5) 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 the runtime.

However, there are many researchers (in particular, research that uses an explicit context model) address the context modelling requirements as shown in Table 1, most of these techniques are not fully address our requirements. Therefore, a rich context model is needed to (1) capture all aspects of the system context (e.g. problem, operational, solution, physical and interaction oriented context, and etc.); (2) have a runtime representation that can be changed while the system is in operation to cope with unexpected environment changes.

6.2 The System Modeling

Modelling aspects: The system needs to adapt its structure and/or behaviour in response to context change. Current research in self-adaptivity models the system behaviour for system verification [23, 29] or the system structure for enabling its construction [23, 35, 28, 30]. In the other hand, the research in context awareness concentrates in modelling the system context without pay attention for modelling the system structure or its behaviour [31-33]. Therefore, there is a need to model the system structure and its behaviour and maintain them during the runtime, which they are adaptable to
cope with context/requirements changes. Furthermore, in case of maintaining multiple system models, they should be consistent with each other, where any change in one model need to be reflected into the other model.

In response to the context change, the system adapts itself. This adaptation should be performed during the runtime while preserving the system requirements. The system requirements are two types: (1) functional requirements which can be represented as the goals model, and (2) non-functional requirements, which can be represented as performance, reliability, and security models. Therefore, these models need to be maintained at runtime and checked with the system model after applying the adaptations actions to assure the system correctness.

**Model management:** Not only there is need to have a consistent system structure and behaviour together with the quality models, but also the other requirements of the system modelling need to be consider for improving its usability and effectiveness such as (1) a tool support for enabling system designing and its construction and put forward a runtime representation of the system model; (2) a formal representation to enable system correctness check and its consistency with the context model.

### 6.3 The System and its Context Relationship

**Monitoring:** In order to monitor the system and its context, their entities must provide the ability to be monitored by giving their status. Current research uses one of the following monitoring methods: continuous [23, 25, 28, 29, 32, 33], event/notification [30], and combination of them [31, 34, 40]. Most of the proposed techniques use continues monitoring where it provides a correct context/system state, but it has a large overhead. In the other hand, one technique has used event/notification which the correctness of the system/context state can’t be guaranteed. The future direction is to use combination of them by making a tradeoff between the monitoring overhead and the correctness of the system/context state. Furthermore, low power devices have limited resources; therefore an efficient mechanism with adaptable context awareness that changes the set of monitoring elements based on the system requirements is needed to reduce the monitoring overhead.

**Adaptation:** Current research adaptation mechanisms to cope with context change are classified into three different types as follow: (1) *action policies*, where in the condition of context change the system will change itself based on what is specified in action part of adaptation rules [30, 35-37]; (2) *goal policies*, where the system behavior is modeled using a state based model like Labeled Transition System (LTS) or Petri Nets (PN). In the case of context change, set of high level goals is used together with the system model to identify the required adaptation actions. These actions moves the system from one state to another (i.e. from system behavior/structure to another) [23, 29]; (3) *utility polices*, where context change will make the system evaluate its alternative configurations based on a set of utility functions and then select the configuration with the best utility [25, 28]. Adaptation rules are expressive and easy to write, but are prone to error (i.e. rules conflict). In the other hand building LTS, PN, or utility functions is difficult but it can be verified for correctness. Therefore, an approach that combines them with each other is needed where it is easy to build and can be verified to assure its correctness.

Number of adaptations is based on the size of the context variables and their variations, and then in large scale system with complex environment, there are a huge number of adaptations. It is difficult to consider a large number of system variations at design time, so how to consider a large number of adaptation behaviors need to be addressed in modeling the system and its context relationship. This problem can be solved by providing an online planner which it is automated or semi-automated to generate a new configuration on-the-fly at runtime. To ensure the correctness of the system when this new configuration will be applied, runtime verification is needed. This poses the challenge of how to
perform fast automated runtime verification. Furthermore, the adaptation policies mechanism should have the ability to be transformed to a code automatically.

**Acting:** Adaptation actions need to be performed at runtime to keep the system in operation as much as possible, then online dynamic execution of operations is required rather than offline maintenance or version control operations. 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 change in decentralized manner is a major challenge.

In vehicle/mobile systems, the context changes are frequent and may be every second and it may needs a response from the system to adapt itself. In these systems not all adaptation actions have the same importance (e.g. safety versus navigation based adaptations in vehicle system), then priority should be considered in selecting the adaptation actions. Also, when there are multiple adaptation actions are pending, this poses a question of how to stop a running adaptation to perform another one with a higher priority and need to be performed immediately. In case of stopping adaptation, rollback its changes is required to ensure system consistency. Furthermore, these systems are running in low powered devices, therefore adaptation overhead versus its benefits need to be considered.

### 7 Conclusion

Current research in context awareness and self-adaptivity are treated separately, but in real world there is a need to combine them with each other as shown in our motivating scenario. In this research report, we have described our vision of context aware adaptive software systems from the system and its context relationship perspective.

We started by setting up the requirements that are posed integrating the context awareness and self-adaptivity. These requirements are partitioned into three categories (1) context modelling; (2) system modelling; (3) the system and its context relationship, to provide a context for analysing context aware adaptive systems and discussing the main research challenges they poses. The main challenge at the context modelling is how to have a context modelling technique that captures all the system environment aspects. Maintaining adaptable multiple system models (i.e. the structure, behaviour and quality models) and maintain their consistency with each other at runtime is a major challenge in the system modelling. System and its context relationship challenge is how to (1) have an efficient monitoring technique; (3) adaptation mechanism that handle large number of adaptations; (3) safely apply the adaptation action; (4) schedule multiple concurrent adaptations.

To provide a context aware adaptive software system, solutions to these challenges need to be 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 vehicles system, to see if these solutions are valid or not.
References


18. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading, Massachusetts, 1995.


