

Situation-Aware Based Self-adaptive Architecture for Mission Critical Systems

Sangsoo Kim, Jiyong Park, Heeseo Chae, and Hoh Peter In*

College of Information & Communications, Korea University,
5-ga, Anam-dong, Seogbuk-gu, Seoul 136-701, Korea
{sookim, jayyp, royalhs, hoh_in}@korea.ac.kr

Abstract. Conventional mission-critical systems cannot prevent mission failure in dynamic battlefield environments in which the execution situations or missions change abruptly. To solve this problem, self-adaptive systems have been proposed in the literature. However, the previous studies do not offer specifics on how to identify changes in a system situation or to transform situation information into the actions the systems must take in dynamic environments. This paper proposes a situation-awareness based self-adaptive system architecture (SASA) to support more efficient adaptation and, hence, achieve more accurate and successful missions, even in dynamic execution environments. A case study for air defense systems (ADS) using tests in a HLA/TRI-based real-time distributed simulation environment was implemented.

Keywords: Situation-Awareness, Self-Adaptation, Mission Critical System, HLA/RTI, Real-Time Distributed Simulation.

1 Introduction

Mission critical systems are somewhat limited in certain fields, as the name implies [1]. Therefore, their applications are also usually limited and fixed to the early assumptions of their operating circumstances and goals. To solve this problem, self-adaptive middleware concepts in designing system architecture have been proposed in the literature [2][3]. However, previous studies have not addressed specific ways to recognize changes in system environments and situations or to transform the situation information into the response actions the systems must execute.

In the real world, mission-critical real-time systems are often exposed to complex, dynamic situations, especially in ubiquitous battlefield environments. Therefore, it is inevitable that much information is generated about the systems and this information affects how efficiently the systems run. Thus, an adaptive architecture is required to reflect the dynamically generated, specific information in the systems. In particular, systems in the military domain can be affected by such dynamic environments more than systems in other domains. Battlefield

* Corresponding author.

environments have changed dynamically in the past and will continue to change dynamically in the future.

In the future, even the near term, new designs in aircraft and missiles that can render air defense systems (ADS) useless will be developed for use on the battlefield. Therefore, to guarantee and maintain the defense capability of ADS from this threat, ADS should be able to understand the actions and situation of an enemy aircraft or missile and destroy it by adjusting a weapon system's search and track algorithm to each situation. Determining a strategy suitable to each situation and changing the guidance algorithm for missile control should be achieved in a very short time to allow immediate response actions. It is difficult for ADS to respond quickly if control commands must go through a ground control center (GCC). Common air defense systems use the conditional expression method, one of the self-adaptive categories, for defense. Nevertheless, this method is still relatively inadequate for bringing threats under control. Therefore, a system that automatically recognizes any changes in environment, evaluates its functional situation awareness, establishes a suitable strategy, verifies the strategy's effectiveness, and executes the mission in real time would be indispensable.

In this paper, self-adaptive system architecture (SASA) for supporting ADS which is now being testing for enhanced future ADS systems is proposed. For better adaptation, situation-aware middleware is applied to our proposed software architecture. The proposed SASA consists of three parts: situation-aware middleware, self-adaptive software, and real-time systems. To verify the effectiveness of the proposed architecture, we tested it in the HLA/RTI based real-time distributed simulation environment now widely used for various domains, including the testing and evaluation of Department of Defense (DoD) weapons systems [4].

The related works which are the basis of our proposed architecture are introduced briefly in chapter 2, and the self-adaptive system architecture (SASA) for mission critical real-time systems, especially ADS, are presented in chapter 3. In chapter 4, the results of implementing the HLA/RTI-based real-time distributed simulation for ADS-adapted SASA are presented, and chapter 5 describes the testing by simulation federation implemented to verify the effectiveness of the self-adaptive requirements. Finally, we present our conclusions and discuss future work.

2 Related Work

The research in air defense systems (ADS) in the military domain is conducted on an ongoing basis. ADS are composed of distributed real-time systems because they collect information and manage numerous communication systems. The problems of scheduling, load balancing, and coordination among the nodes of distributed real-time systems are studied in [5]. ADS can reduce inconsistent, unintegrated, and inexact information and actively deal with a mission if a knowledge-based database is composed because a knowledge system can more efficiently cope with emergencies. The system in [6] offers a decision-making

system that uses a war control center. In addition, the research in [7] develops a situation analysis system and estimates a danger state using fuzzy logic technologies. However, it is difficult to adapt for mission change that occurred during the mission time, though those methods are aptitude to ascendant initial target.

Various types of research have been done about self-adaptive systems, such as an architecture-based approach [3], rainbow [2], self-optimization [8], a self-adaptation genetic algorithm [9], and self-adaptive for robotics [10]. Most of this research focused not on real-time systems but on everyday systems and non-mission critical systems. Self-adaptive systems for real-time methods were studied by [11]. However, most researchers focused on the adaptive methodology and algorithm.

In a ubiquitous environment, situation-aware middleware and language for specifying situation awareness is researched. In [12], reconfigurable context-sensitive middleware is extended in ubiquitous computing environments to simplify the development of Situation-Awareness (SA) application software and achieve reusability and runtime reconfigurability.

In this paper, we apply situation-aware middleware, which is briefly introduced in the next section, to our proposed situation-aware based self-adaptive system architecture (SASA) for accurate information analysis about mission environments. Situation awareness can significantly improve the performance of ADS. Typical real-time systems and knowledge-based systems have insufficient flexibility and treating capacity because of lack of data. In an unpredictable situation, the systems will have difficultly managing emergencies and will demonstrate inferior agility because they must communicate through a centralized system and process large amounts of general data. System-based situation awareness can overcome these defects.

3 Situation-Aware Self-Adaptive System

3.1 SASA for ADS

A self-adaptive system architecture (SASA) for ADS has the form of a missile system complemented by self-adaptive systems properties. SASA has been adapted to guided missiles among ADS subsystems. As Fig. 1 shows, the missile-adapted SASA consists of three parts.

Situation-Awareness Expresser (SAE): Analyzes situations and derives actions from radar, seeker, GPS, and other external information. It collects situation data in a database and receives the required context acquisition. The situation analysis feature determines the present state based on the data collected, then assigns and schedules action tuples.

Adaptive Engagement Manager (AEM): If a new strategy is required after the extracted actions are compared with an existing engagement strategy, the new strategy is produced by the adaptation engine.

Guidance and Control Unit (GCU): The new strategy is passed through a self-adaptation executer to the weapons system's guidance and control unit (GCU).

- *Action Scheduler*: Identifies the actions to be triggered by the recognized situations, approves permitted actions, and schedules these actions.
- *Action Tuple*: $\langle t, a1, a2, \dots, an \rangle$, where t is the time the action is performed and $a1, a2, \dots, an$ is a set of attributes of the action. Attributes can be name, parameter types, parameter values, and the like. When an action is taken, information about the action is recorded using an action tuple value.
- *Situation Expression*: Has the format: $[\forall, \exists] t \text{ in } \langle \text{time range} \rangle [\text{context}, \text{derivative}, \text{action}, \text{situation}] \langle \text{compare} \rangle \langle \text{value} \rangle$.

We can form new situations by performing "not," "and," or "or" operations in defined situations. To specify an event-handling plan, i.e., what actions should be taken to respond to different situations, we define related rules to associate actions to situations as follows: $[\text{activate at situation } x] \text{ action } y \dots$ A list of such "situation x " and "action y " forms a plan. The event handler monitors the situation-match events, and activates the actions associated to them according to the plan.

3.3 Situation-Aware IDL

In a mission situation, SASA decides new missions that change the original targets based on the priority of the target type and distance between the target and missile. In order to suggest the development method which can describe the situation and make the situation-aware service, SA-IDL is used. The SA-IDL model such a tuple format is essential to justify the collaborative relation and conflict. Thus SA-IDL model has the component which are Context Tuple, Action Tuple and Derived Context as follows:

$$\begin{aligned} \text{Action} &:= (\text{Time}, \text{Target}, \text{State}, \text{StateValue}) \\ \text{Context} &:= (\text{Timestamp}, \text{Target}, \text{State}, \text{StateValue}) \\ \text{A} &:= \{x \mid x \text{ is an Action}\} \\ \text{C} &:= \{x \mid x \text{ is a Context}\} \\ \text{DerivedContext} &:= P(\text{C}) \longrightarrow \{\text{true}, \text{false}\} \\ \text{Situation} &:= (\text{DerivedContext}, P(\text{C}), \text{A}) \end{aligned}$$

In action tuple, an action is used in two manners: one is to change attribute values of target, and the other is to give orders to target. The context has information of target each time, and the situation constitutes the "Derived Context" that accumulated the information of the context. Especially, the state of action tuple and context tuple have attributes of attitude, speed, coordinates, priority of target, and the "StateValue" is the real value of the states.

The system operates situation awareness in a target with the knapsack algorithm [14], which is the one of dynamic programming and mathematical models that defined the successful condition of ADS. The system can reduce total mission time by situation awareness.

Fig. 2 shows a portion of the SA-IDL specification for the application software based on the identified situation-awareness requirements. For example, the action

```

SASAApplication SA {
Class Time {int year; int month; int day; int hour; int minute; int second; int millisecond; int microsecond;}
Class Location {String loc;}
Class History {Time beginTime; Time endTime; List string idList;}
Class Mission {String ID; Time beginTime; Time endTime; List string enemyList;}
Class Enemy {String ID; String priority; String type; String speed; String altitude;}
    Context Collection Rate = 0.1;
    ...
    CompositeSituation readyToMission (Mission m, Enemy en, Histroy h)
        collectSituation(m, t, loc) && allowToMission(m, en, h)
    CompositeSituation emergencyState (Mission m, Enemy en, Histroy h)
        immediateState(m, t, h) && alarm(m) && checkMission(m, en, loc)
    ...
    Rule missionState ActiveAt readyToMission(Mission m, Enemy en, Histroy h) {[local]}
void missionState();
    Rule emergency ActiveAt emergencyState(Mission m, Enemy en, Histroy h) {[local]}
void emergency();
    ...
}

```

Fig. 2. Partial SA-IDL Specification

'missionState' is triggered when 'the system collects the start of mission caused by enemy' (*collectionSituation*) and 'the center system is allowed to start the mission' (*allowToMission*). The action 'emergency' is triggered by 'the mission starts with unpredictable situation' (*emergencyState*).

Definition of success and selection of new definition of SA-IDL through modeling is presented as shown in Fig. 3.

<pre> Success:=(C, Action, Action) { ∃ (c, x, y): Success; ∀ a, b: Situation c ∈ a.P(C) ∩ b.P(C) and c ≠ ∅ and a.DerivedContext (c) = true and b.DerivedContext (c) = true and x ∈ a.A and y ∈ b.A and x.Time = y.Time and x.Target = y.Target and x.State = y.State and x.NewState ≠ y.NewState } </pre>	<pre> Select:=(C, Action, Action) { ∃ (c, x, y): Select; ∀ a, b: Situation c ∈ a.P(C) ∩ b.P(C) and c ≠ ∅ and a.DerivedContext (c) = true and b.DerivedContext (c) = true and x ∈ a.A and y ∈ b.A and x.State = y.State and x.Priority > y. Priority and x.Target ≠ y.Target and x.NewState ≠ y.NewState then Success(C, Action, Action)} </pre>
---	--

Fig. 3. Definition of Success and Select

There exists a conflict (c, x, y) for any two situations, a and b. The above constraint defines the context set c that triggers successful action. The intersection of $P(a.C)$ and $P(b.C)$ denotes a set of context sets that is shared between two situations, a and b. The context set c should be an element of this intersected set. Also, it is an input element of the Derived Context functions, and it satisfies both requirements of the two situations. Additionally, the c should not be an empty set. The above constraint defines two actions, x and y , that are same as

each other. X is an element of a 's action set, and y is of b 's. The two actions change the same state of the same target at the same time with different values.

However, the mechanism of target selection with the success of ADS has different forms in the priority. The priority of a new target is different from that of the original target; and then the missile selects a new target to define of a successful situation.

4 Implementing the SASA

4.1 ADS Simulation

We implemented ADS to verify the effectiveness of our SASA. To implement and test it in conditions most similar to real systems, we implemented the HLA/RTI-based real-time distributed simulation. HLA/RTI is now applied by the DoD Defense Modeling and Simulation Office (DMSO) for simulation-based weapons systems testing and evaluation. Thus, it is an infrastructure suitable for virtual experiments under the circumstances of real-time systems.

ADS have been developed to assess the performance of runtime infrastructure (RTI) for high-level architecture (HLA) and to test or evaluate surface-to-air missiles now under construction [4][15]. Through this paper, we present our remodeled ADS, which has SASA applied, and verify its effectiveness. ADS simulation has seven components: Simulation control center (SMCC), air target simulator (ATS), multifunction radar (MFR), engagement control simulator (ECS), launcher (LAU), missile (MSL), and runtime infrastructure (RTI).

The SMCC monitors and controls the real-time distributed simulation for air defense systems. The ATS creates the air targets and transmits their information. The MFR searches out the position of airborne targets and missiles. The ECS evaluates the threat degree of any airborne targets and assigns threat priority to the missiles. The LAU transfers information from the ECS to the MSL and simulates the launcher function. MSL represents the missile and traces the airborne targets by threat evaluation. These six components interact with each other, guided by the RTI.

4.2 Algorithm for Engagements

To achieve a mission in a various and changeable situation, SASA needs an algorithm that can perceive a situation and threat it actively. The algorithm for engagements in mission-critical situations is given below:

- Step 1. The ADS scans the situation in the radar capable region and updates the situation data. It checks for changes in the general situation or allocation mission and sends the data to the SA Data Manager.
- Step 2. The mission must be check-and-change adaptable in the present situation in the mission state if new mission data about the number of targets and acquisition of new targets is not to coincide with the general situation.

- (a) The situation that finds unpredictable target.
- (b) The situation that makes a change in a target.
- Step 3. The ADS terminates the mission situation and updates the situation data in the database.

In Step 2(a), the system needs a priority algorithm for each situation. Military aircraft allocation algorithms are applied to the field of large-scale allocation problems in which a collection of resources (assets) must be mapped in an optimal or near-optimal manner to a number of objectives (targets), as measured by an objective function [16].

$$\Pi_j = 1 - \prod_{i=1}^m (1 - x_{i,j} p_{i,j}) \tag{1}$$

$$U(X) = \sum_{j=1}^n P_j \Pi_j \tag{2}$$

$$U(X) = \sum_{i_1=1}^m \sum_{i_2=1}^m \sum_{j=1}^n \varepsilon_{i_1, i_2, j} x_{i_1, j} x_{i_2, j} \tag{3}$$

$$Y(X) = \sum_i v_i \sum_j \gamma_{i,j} x_{i,j} \tag{4}$$

If the efficiency $p_{i,j}$ is interpreted as the probability of the event "elementary asset i will achieve elementary objective j if so allocated," and if these events are statistically independent, then the probability of elementary objective j being achieved by allocation strategy X is (2). The degree to which the events fail to be independent is captured by $V(X)$, which quantifies the additional benefit of allocating multiple assets to the same objective. $Y(X)$ measures the cost of the strategy X . This cost includes deterministic costs, such as fuel and ordnance, as well as statistical costs, such as risk of asset damage or loss. Combining these terms (2), (3), (4), gives us the final algorithm in (5):

$$J(X) = AU(X) + BV(X) - rY(X) \tag{5}$$

where, the efficiency $p_{i,j}$ of asset B_i in achieving objective T_j , the value v_i of asset B_i , the risk $\gamma_{i,j}$ associated with allocating B_i to T_j , and the joint efficiency $\varepsilon_{i_1, i_2, j}$ added by simultaneously assigning assets B_{i_1} and B_{i_2} to objective T_j .

The SASA system controls an objective with a high-priority target using the algorithm in each situation. After the mission, it updates the situation data in the database and refers to it in next mission.

5 Evaluation

5.1 Test Environments

The test results, which include performance measurements, gave us information about the mission elapsed time and the hit rate for missiles. We validated

the usefulness of the systems having self-adaptation capability as a result, then compared systems having this capability with systems without it.

The mission elapsed time is the period of time from missile launch to target shoot-down to accomplish a mission. It is calculated as follow:

$$T_{mission} = T_{hit} - T_{launch} \quad (6)$$

Where $T_{mission}$ is the mission time, T_{hit} is a time when the missile hits the target or explodes by itself, T_{launch} is when the missile is launched to engage the target.

The hit rate of missiles measures the success rate in shooting down targets. The probability of single shot kill (P_{ssk}) is used as the hit rate. Assume, if given one shot,

$$P_{ssk} = \int_{-\infty}^{\infty} f(x) \cdot l(x) \cdot d(x) \quad (7)$$

Where $f(x)$ is hit probability, $l(x)$ is lethality function. The Gaussian lethality function is:

$$l(r) = P_0 \cdot e^{\frac{-x^2}{2a^2}} \quad (8)$$

Where P_0 is kill probability at $x=0$ (x is the distance from the target) and lethality constant a is predefined as weapon characteristics. It is assumed that our missile's lethality constant is 7 meters.

In the execution environments, we changed the number of targets and missiles to represent dynamic mission situations. The test outputs allowed us to distinguish between the results for systems having self-adaptation capability and systems without it.

5.2 Test Results

As Table 1 shows, we obtained average values by comparing the systems having situation-aware based self-adaptation capability with the systems without it, judged according to the number of targets. Systems 1 and system 2 are, respectively, the first and second targets in the mission simulation.

According to the test results, the mission time of the systems having situation-aware based self-adaptation capability shows moderate reduction over the elapsed time of systems without this capability. Within a 10ms error rate, the results reveal

Table 1. Mission Time of non-SASA and SASA

Number of Target	Non SASA		SASA	
	System 1	System 2	System 1	System 2
4	36804.91	38920.95	36736.03	38846.31
6	46504.26	49170.48	46469.68	49035.68
8	48927.71	51800.27	48422.40	51621.94
10	57340.85	60808.92	56773.07	60120.36

a gap in elapsed time between the situation-aware self-adaptive systems and the other systems as the number of targets increases. The results also show a maximum term of 680ms. The reason for this is that a missile target has changed without commands from the GCC.

Table 2. Accuracy Rate of SASA and Non-SASA

Number of Target	Non SASA	SASA
4	87	90
6	86	88
8	84	88
10	84	88
Average	85.25	88.5

Table 2 shows the accuracy rate of the SASA applied system and the non-SASA applied system in conditions in which the targets are changed. Because ADS have demonstrated an 85% mission success rate in real-world situations, we can assume that the non-SASA systems in this paper also have an 85.25% success rate. The approximately 15% mission failure rate is caused by defects in a missile, incorrect radar commands, missile control errors, and tracking faults caused by changes in targets. In the SASA applied systems, the systems estimate an 88.5% mission success rate in the simulation because SASA can reduce the mission elapsed time to shoot down and destroy targets while continuing to track the most threatful and dynamic targets in the absence of commands from the MFR.

6 Conclusions

This paper presents a self-adaptive system architecture (SASA) for surface-to-air missile guidance. To achieve a more sophisticated adaptation capability, we applied situation-aware middleware to our work. The SASA presented in this paper includes compositions of concepts such as situation awareness, self-adaptation, verification, and real-time systems. To verify the proposed architecture, we implemented a HLA/RTI-based real-time distributed simulation, tested it, and compared the systems having self-adaptation capability with systems lacking that capability. According to the test results, the time consumed in processing self-adaptation during a mission did not influence mission success.

On the contrary, the total mission execution time was shortened by saving the communication time between the missile and the GCC, a conventional situation. The self-adaptive system determined the mission properly by referring to changes in the surrounding situation and finally processed the mission in valid time. Although the communication delay time between subsystems eventually increased, we could confirm that the mission success rate was considerably increased. Consequently, the SASA adapted system not only saved on total mission execution time but also increased the hit rate of missiles. It is expected that our

proposed architecture will output equivalent performance in real systems because the simulation and test environments are based on HLA/RTI, which is basically used for weapons systems testing and evaluation by the DoD.

As future work, we plan a study to verify the self-adaptive systems of real surface-to-air missile systems and check the effectiveness of an engagement strategy automatically determined by these self-adaptive systems through a model checking approach. In addition, situation awareness expression language is necessary for hard, real-time systems in a mission critical case.

Acknowledgments. This work was supported by the 2nd Brain Korea 21 Project in 2006.

References

1. Krishna, C. M., Shin, K. G.: Real-Time Systems, McGraw Hill, New York (1999)
2. Garlan, D., Cheng, S., Huang, A., Schmerl, B., Steenkiste, P.: Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure, *IEEE Computer*. October (2004) 46–54
3. Oreizy P., Gorlick, M. M., Taylor, R. N., Heimbigner, D., Johnson, G., Medvidovic, N., Quilici, A., Rosenblum, D. S., and Wolf, A. L.: An Architecture-Based Approach to Self-Adaptive Software, *IEEE Intelligent Systems*. May/June (1999) 54–62
4. Lee, T. D., Jeon, B. J., and Choi, S. Y.: RISA: Object-oriented Modeling and Simulation of Real-time dIstributed Systems for Air defense, *Lecture Notes in Computer Science*, OOIS'03. September (2003) 346–355
5. Choi, S. Y., Wijesekera, D.: The DADSim: Air Defense Simulation Environment, *High Assurance Systems Engineering*, Fifth IEEE International Symposium on. HASE (2000) 75–82
6. Lin, C. E., Chen, K. L.: Automated Air Defense System Using Knowledge-Based System, *IEEE transactions on aerospace and electronic systems*. v.27 no.1 (1991) 118–124
7. Hua, X. Q., Jie, L. Y., Xian, L. F.: Study on Knowledge Processing Techniques in Air Defense Operation Intelligent Aid Decision, *Computational Intelligence and Multimedia Applications*, ICCIMA 2003. Proceedings. Fifth International Conference (2003) 114–119
8. Ganak, A. G., Corbi, T. A.: The Drawing Automatic Computing era. *IBM System Journal* v.42 no.1 (2003) 5–18
9. Wang, Z., Cui, D., Huang, D., and Zhou, H.: A Self-Adaptation Genetic Algorithm Based on Knowledge and Its Application, *Proceedings of the 5th World Congress on Intelligent Control, and Automation*. June 15-19, Hangzhou, P.R. China (2004) 2082-2085
10. Kim, J., Park, S.: Self Adaptive Software Technology for Robotics, *Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC'04)* (2004)
11. Shetty, S., Neema, S., Bapty, T.: Model Based Self Adaptive Behavior Language for Large Scale Real Time Embedded Systems, *IEEE Conference on the Engineering of Computer Based Systems (ECBS)*, Brno. Czech Republic. May (2004)
12. Yau, S., Huang, D., Gong, H., Seth, S.: Development and Runtime Support for Situation-Aware Application Software in Ubiquitous Computing Environments, *COMPSAC -NEW YORK-*, v.28 (2004) 452–457

13. Yau, S., Wang, Y., and Huang, D., In, H.: Situation-Aware Contract Specification Language for Middleware for Ubiquitous Computing, The Ninth IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS'03) (2003) 93–99
14. Martello, S., Toth, P.: Knapsack Problems: Algorithms and Computer Implementations, John Wiley & Sons, New York (1990)
15. Jeong, C. S., and Choi, S. Y.: An Object-oriented Simulation Systems for Air Defense, Lecture Notes in Computer Science. May (2003) 674–683
16. Abrahams, P., Balart, R., Byrnes, J. S., Cochran, D., Larkin, M. J., Moran, W., Ostheimer, G.: MAAAP: the Military Aircraft Allocation Planner”, Evolutionary Computation Proceedings, IEEE World Congress on Computational Intelligence, The 1998 IEEE International Conference (1998) 336-341