Development and Maintenance Environment for Anticipatory Reasoning-Reacting Systems

Yuichi Goto, Ryota Kuboniwa, and Jingde Cheng
Department of Information and Computer Sciences, Saitama University
Saitama,338-8570, Japan
{gotoh, kuboniwa, cheng}@aise.ics.saitama-u.ac.jp

Abstract
Anticipatory reasoning-reacting systems were proposed as a new generation of reactive systems with high reliability and high security. At present, there is no engineering environment to support development and maintenance of anticipatory reasoning-reacting systems. This paper presents a development and maintenance environment we are building for developing and maintaining anticipatory reasoning-reacting systems. By using our development and maintenance environment, developers and maintainers can develop and maintain various practical anticipatory reasoning-reacting systems more easily.
Keywords : Anticipatory reasoning-reacting systems, Predictor, Decision-maker, Persistent computing systems, Engineering environment.

1 Introduction

Anticipatory reasoning-reacting systems (ARRSs) were proposed as a new generation of reactive systems with high reliability and high security [1]. An ARRS predicts possible failures and attacks by detecting their omens and anticipatory reasoning about failures and attacks based on logic systems, empirical knowledge and detected omens, informs its users about possible failures and attacks, and performs some operations to defend the system from possible failures and attacks anticipatorily by itself.

At present, there is no engineering environment to support development and maintenance of anticipatory reasoning-reacting systems. Such an engineering environment is necessary. To develop and maintain a practical anticipatory reasoning-reacting system is not an easy task because an anticipatory reasoning-reacting system consists of many components and it must work based on some fragments of logic systems. On the other hand, although different anticipatory reasoning-reacting systems may include different functional components and may work based on different logic systems, there are certainly some common functional/non-functional components and some fragments of logic systems that may be developed and maintained in a way independent of applications. There were several studies for ARRSs [1, 2, 6, 7, 11, 12, 18, 19, 22, 23, 24, 25, 28], but there is no study for the engineering environment, yet.

This paper presents a development and maintenance environment we are building for developing and maintaining anticipatory reasoning-reacting systems. The paper identifies and classifies common functional/non-functional components in all anticipatory reasoning-reacting systems, gives a requirement analysis for the development and maintenance environment, and presents some common components we are developing for various anticipatory reasoning-reacting systems.

2 Earlier Studies of Anticipatory Reasoning-Reacting Systems

The one of the most important facilities for anticipatory reasoning-reacting systems is a facility of anticipation. *Anticipation* is the action of taking into possession of some thing or things beforehand, or acting in advance so as preclude the action of another. It is a notion must relate to two parties such that the party taking anticipation acts in advance of a proper time earlier than the time when another party acts. To implement the facility of anticipation, we can naturally find following three issues: 1) how to predict future event or events, 2) how to take next actions, and 3) how to ensure that a system behaves continuously and persistently without stopping its running. Earlier studies tackled those issues.

As a methodology of prediction, a method using anticipatory reasoning based on temporal relevant logics or 3D spatio-temporal relevant logics was proposed [2, 11]. *Prediction* is the action to make some future events known in advance, especially on the basis of special knowledge. It is a notion must relate to point of time to be considered as the reference time. For any prediction, both the predicted thing and its truth must be unknown before the completion of that prediction. An *anticipatory reasoning* is a reasoning to draw new, previously unknown and/or unrecognized conclusions about some future event or events whose occurrence and truth are uncertain at the point of time when the reasoning is being performed [2]. To represent, specify, verify and reason about various objects in the real world and relationships among them in the future, any ARRS needs a right fundamental logic system to provide a criterion of logical validity for anticipatory reasoning as well as formal representation and specification language. Temporal relevant logics and 3D spatio-temporal relevant logics are hopeful candidates of such right fundamental logic systems for ARRSs [2, 11]. Furthermore, to perform anticipatory reasoning automatically, an anticipatory reasoning engine was proposed and its prototype was implemented [12, 18, 25]. An anticipatory reasoning engine is a forward reasoning engine to perform anticipatory reasoning based on a fragment of temporal relevant logics or 3D spatio-temporal relevant logics. A forward reasoning engine is a computer program to automatically draw new conclusions by repeatedly applying inference rules to given premises and obtained conclusions until some previously specified conditions are satisfied. A fragment of a logic system L is a finite subset of logical theorems of L. Moreover, a prototype of ARRS for elevator control was implemented, and the prototype showed the prediction method is useful to implement a facility of prediction in anticipatory reasoning-reacting systems [28].

On the other hand, a decision-making method with reasoning about actions was proposed [22, 23, 24]. *An action* in a computing anticipatory system is a deed performed by the system such that as a result of its functioning a certain change of state occurs in the system. To take next actions, at first, a computing anticipatory system enumerates all actions that the system can perform in a predicted future situation as candidates of next actions, and then, the system chooses appropriate actions as next actions to defend the system from possible failures and attacks. *Reasoning about actions* in a computing anticipatory system is the process to draw new conclusions about actions in the system from some given premises, which are already known facts or previously assumed hypotheses concerning states of the system and its external environment [23]. The decision-making method uses reasoning about actions to enumerate candidates of next actions. Deontic relevant logics and temporal deontic relevant logics are adopted as hopeful candidates of right fundamental logic systems for reasoning about actions [6, 22, 23]. Furthermore, to perform reasoning about actions automatically, an action reasoning engine was proposed and its prototype was implemented [22, 23]. Like the anticipatory reasoning engine, an action reasoning engine is a forward reasoning engine to perform reasoning about actions based on a fragment of deontic relevant logics or temporal deontic relevant logics. Moreover, a prototype of ARRS for terminal radar control was implemented, and the prototype showed the decision-making method is useful to implement a facility of decision-making in ARRSs [24].

By the way, the notion of anticipatory system [27], in particular, computing anticipatory system [14, 15, 16], implies a fundamental assumption or requirement, i.e., to be anticipatory, a computing system must behave continuously and persistently without stopping its running, because (1) for any anticipatory system, concerning its current state, there must be a future state referred by the current state, and (2) for any anticipatory system, its states form an infinite sequence [7]. Thus, Cheng and Shang showed that persistent computing systems should be as an infrastructure of computing anticipatory systems [7]. A persistent computing system is a reactive system that functions continuously anytime without stopping its reactions even when it is being maintained, upgraded, or reconfigured, it had some trouble, or it is being attacked [3, 4, 5]. Conceptually, a reactive system is a computing system that maintains an ongoing interaction with its environment, as opposed to computing some final value on termination [20, 26].

3 PCS-core Components and ARRS-core Components

Earlier studies for ARRSs made clear how to predict future event or events, how to take next actions, and how to ensure that a system behaves continuously and persistently without stopping its running. We, therefore, re-defined an architecture of an ARRS according to results of the earlier studies. Figure 1 shows an architecture of an ARRS.

An ARRS is a persistent computing system. A persistent computing system can be constructed by a group of control components including self-measuring, self-monitoring, and self-controlling components with general-purpose that are independent of systems, a group of functional components to carry out special tasks of the system, some data-instruction buffers, and some data-instruction buses [7]. Control components may include a central controller/scheduler (C/S), a central measurer (Me), a central recorder (Rec), a

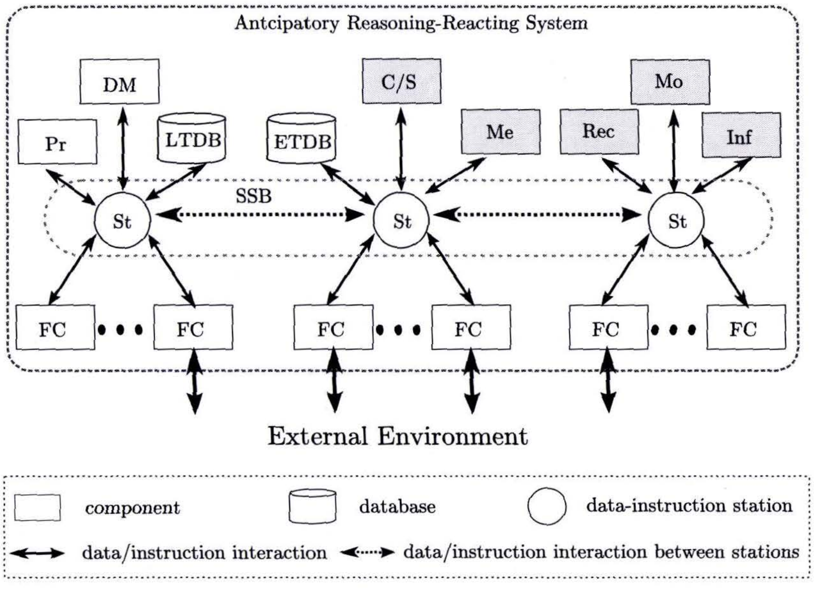
Antcipatory Reasoning-Reacting System
DM
C/S
Mo
Pr
LTDB
ETDB
Me
Rec
Inf
SSB
St
St
St
FC
FC
FC
FC
FC
FC
External Environment
component
database
data-instruction station
data/instruction interaction
data/instruction interaction between stations

Fig. 1: An architecture of an anticipatory reasoning-reacting system

central monitor (Mo), and an central informant (Inf). A central controller/scheduler orders and controls all components to carry out some operations with a high priority. A central measurer measures current status of the system, and stores measured data into a central recorder. A central recorder stores data observed by a central measurer, and provides them to a central monitor and a central controller/scheduler. A central monitor monitors the behavior of the whole of the system, and reports unexpected behavior or troubles to a central informant. A central informant receives reports about unexpected behavior or troubles of the system from a central monitor, and informs the reports to managers of the system. A soft system bus (SSB) is simply a communication channel with the facilities of data/instruction transmission and preservation to connect components in a component-based system. It may consist of some data-instruction stations (St's), which have the facility of data/instruction preservation, connected sequentially by transmission channels, both of which are implemented by software techniques, such that over the channels data/instructions can flow among data-instruction stations, and a component tapping to a data-instruction station can send data/instructions to and receive data/instructions from the data-instruction station. SSBs are used for connecting all components such that all data/instructions are sent to target components only through the SSBs and there is no direct interaction that does not invoke the SSBs between any two components.

Functional components of an ARRS are classified into two kinds components; ones are common components in all ARRSs and others are application-dependent components. A predictor (Pr), a decision maker (DM), a logical theorem database (LTDB), and an

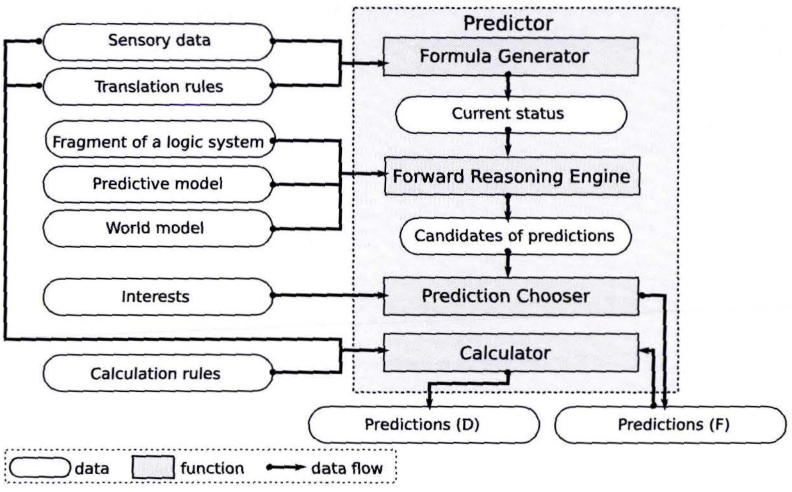

Fig. 2: Data flow diagram of a predictor

empirical theory database (ETDB) are the common components.

A predictor receives several kinds of data and outputs predictions with quantitative information and predictions without quantitative information. Figure 2 shows a data flow diagram of a predictor. A predictor consists of four functions: formula generator, forward reasoning engine, prediction chooser, and calculator. Formula generator takes sensory data and translation rules, and then it translates the sensory data into logical formulas according to the translation rules. Sensory data are data observed by a central measurer or functional components that measure current status of external environments of the system. Translation rules are description rules to make logical formulas without quantitative information from sensory data. Forward reasoning engine gets logical formulas translated at the formula generator, a fragment of a logic system, a predictive model, and a world model, and then it deduces candidates of predictions. A predictive model is a set of empirical theories which are represented by logical formulas and related with time in a target domain of the system. A world model is a set of empirical theories represented by logical formulas in the target domain except empirical theories related with time and behavior. Prediction chooser chooses nontrivial predictions, "Prediction(F)" in Fig. 2 denotes those predictions, from the candidates of predictions according to interests. Interests are selection rules to choose nontrivial predictions. Calculator adds quantitative information to predictions chosen by the prediction chooser according to calculation rules, translation rules, and sensory data. Calculation rules are rules to add quantitative information to predictions chosen by the prediction chooser. The predictions with quantitative information, "Prediction(D)" in Fig. 2 denote predictions with quantitative information, and predictions without quantitative information are sent to a decision-maker.

A decision-maker receives two kinds of predictions from a predictor, and outputs in-

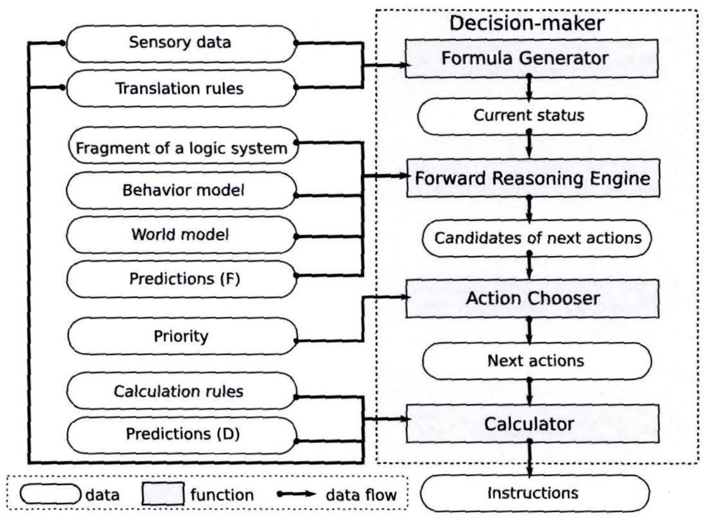

Fig. 3: Data flow diagram of a decision-maker

structions to an ARRS. Figure 3 shows a data flow diagram of a decision-maker. Like a predictor, a decision-maker consists of four functions: formula generator, forward reasoning engine, action chooser, and calculator. Formula generator takes sensory data and translation rules, and then it translates the sensory data into logical formulas according to the translation rules. Forward reasoning engine gets logical formulas translated at the formula generator, a fragment of a logic system, predictions without quantitative information, a behavior model, and a world model, and then it deduces candidates of next actions. A behavior model is a set of empirical theories that are represented by logical formulas and related with behavior in a target domain of the system. Action chooser chooses appropriate actions from the candidates of next actions according to priority. Priority is a set of selection rules to decide next actions. Calculator adds quantitative information to next actions that the action chooser chose by using predictions with quantitative information, calculation rules, translation rules, and sensory data. The next actions with quantitative information are outputted as instructions.

A logical theorem database stores fragments of logic systems underlying anticipatory reasoning or reasoning about actions. An empirical theory database stores empirical theories of a target domain as predictive models, behavior models, or world models.

We identified and classified common functional/non-functional components and application-independent data in all ARRSs. Common components in all ARRSs are classified into two kinds components: common components in all persistent computing systems and others. Control components and soft system buses are common in all persistent computing systems. Hereafter, we name those components *PCS-core components*. A predictor, a decision-maker, a logical theorem database, and an empirical theory database are common components in all anticipatory reasoning-reacting systems, but not in all persistent computing systems. Hereafter, we name those components *ARRS-core components*. The data required by a predictor and a decision-maker are sensory data, fragments of logic systems, a predictive model, a world model, a behavior model, translation rules, calculation rules, interests, and priority. Only fragments of logic systems are application-independent data.

4 Development and Maintenance Environment

A development and maintenance environment for ARRSs is an engineering environment that integrates various tools and provides comprehensive facilities for designers, developers, administrators/end-users, and maintainers of ARRSs. We analyzed and defined basic requirements that the environment should satisfy as follows.

R1: *The environment must provide tools and facilities to support all tasks in design, development, management, and maintenance of ARRSs.* In software life cycle, management and maintenance phases are far longer than design and development phases. Furthermore, ARRSs are persistent computing systems and persistent computing systems should be running consistently and continuously. Maintenance phase, therefore, becomes more important for ARRSs. The environment must provide tools and facilities to support all tasks in not only design and development phases, but also management and maintenance phases.

R2: *The environment must support to ensure the whole security of an ARRS.* ARRSs were proposed as a new generation of reactive systems with high reliability and high security. On the other hand, the whole security of a target system is not necessarily the sum total of security of its all components but usually only as good and strong as the weakest security of some component or link between components in the system. To implement a secure ARRS, the environment must support to ensure the whole security of the system, but not only each component of the system.

R3: *The environment itself must be a persistent computing system to provide continuous supports for all users anytime.* ARRSs should run continuously and persistently until its managers stop it because ARRS are persistent computing systems. The environment, therefore, must support maintenance of ARRSs anytime.

R4: *The environment must support specialists of a target domain of a target ARRS to formulate empirical theories easily and to manage the theories consistently.* To formulate empirical theories in a target domain of a target ARRS is not easy task because most of specialists of the target domain are neither logicians nor programmers. Furthermore, empirical theories may be modified many times because the current situation and priority may be changed. After modification of empirical theories, the set of stored empirical theories in the ARRS may include some contradictions because of human error. Hence, the environment must provide some easy ways to formulate empirical theories, and support to manage stored empirical theories consistently.

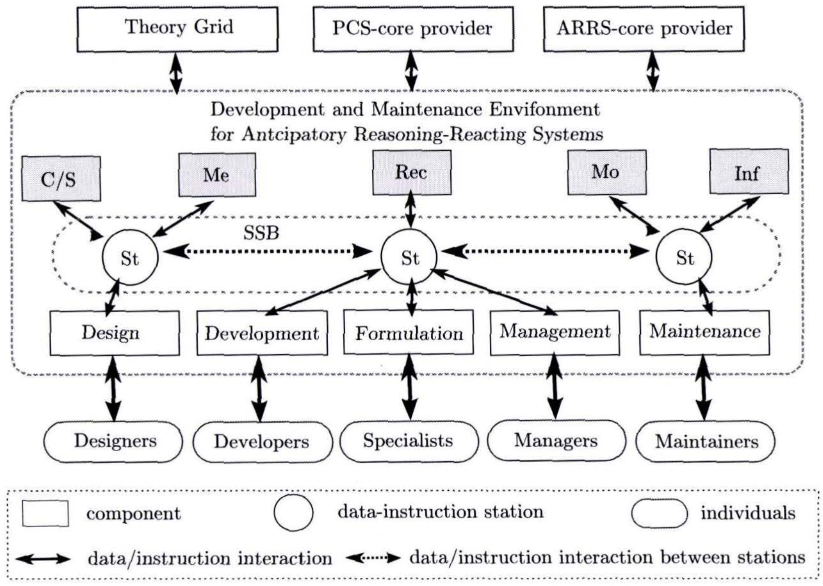

Fig. 4: Structure of a development and maintenance environment

R5: *The environment should provide any fragment of any logic system required by developers of a target ARRS.* As we have mentioned in section 3, fragments of logic systems are application-independent data. It is, therefore, possible to reuse a fragment of a logic system that someone previously prepared. On the other hand, there are several axiomatic systems in temporal relevant logics, 3D spatio-temporal relevant logic systems, deontic relevant logics, and temporal deontic relevant logics. It is possible to generate many kinds of fragments of those logic systems. Moreover, what a suitable logic system/axiomatic system is varies from person to person. We think that temporal relevant logics, 3D spatio-temporal relevant logic systems, deontic relevant logics, and temporal deontic relevant logics are suitable logic systems underlying anticipatory reasoning or reasoning about actions, but someone does not think so. He/she may want to use other logic systems, and prepare those fragments. Thus, it is efficient that fragments of logic systems are shared with all developers of ARRSs.

R6: *The environment must support developers and maintainers of a target ARRS to get the newest PCS-core components and ARRS-core components, and support the maintainers to update those components easily and safely.* As we have mentioned in section 3, it is better that PCS-core components and ARRS-core components are provided from one organization or project like Linux kernel because those components are common in all of ARRSs. In addition, developers and maintainers should adopt the newest version of PCS-components and ARRS-core components at all time because most of facilities of those components satisfy non-functional requirements, e.g., security, reliability, performance, of a target ARRS.

We are developing a development and maintenance environment that satisfies requirements we defined because traditional software engineering environments do not satisfy the requirements [17, 21]. The environment is an engineering environments for ARRSs to provide designers, developers, specialists, managers, and maintainers with standard, formal and consistent supports for the design, development, formulation, management, and maintenance of ARRSs with high security and reliability requirements [8, 9, 10, 13, 19, 29]. Figure 4 shows a structure of the development and maintenance environment.

Our environment is a persistent computing system, and is based on an information security engineering environment (ISEE) [10, 13]. ISEE is an engineering environment that integrates various tools and provides comprehensive facilities for designers, developers, administrators/end-users, and maintainers of information/software systems such that they can use the tools and facilities to ensure the whole security of the target system anytime consistently and continuously according to ISO/IEC security standards. ISEE focuses on security facilities of a target system, so we are modifying ISEE to deal with not only security facilities of a target system but also the system itself. We are also adding support tools for formulating and managing empirical theories into ISEE.

Our environment can get various fragments of various logic systems from the Theory Grid. The Theory Grid is a formal theory infrastructure [9]. It coordinates various fragment of logic systems and formal theories in a distributed way using standard, open, general-purpose protocols and interfaces to meet demands of its application programs for theorem discovery and/or question proposition.

Our environment can get the newest version of PCS-core components and ARRS-core components from a PCS-core provider and an ARRS-core provider, respectively. a PCS-core provider (An ARRS-core provider) provides PCS-core components (ARRS-core components). We are developing a PCS-core component: a structured p2p based SSB [29], and an ARRS-core component: FreeEnCal [8]. FreeEnCal is a forward reasoning engine with general-purpose, and is a hopeful candidate for a forward reasoning engine in a predictor and a decision-maker.

5 Concluding Remarks

This paper identified and classified common functional/non-functional components in all anticipatory reasoning-reacting systems (ARRSs). The components are classified into two kinds components: common components in all persistent computing systems and others. The former are control components and soft system buses, and the latter are a predictor, a decision-maker, a logical theorem database, and an empirical theory database. Application-independent data is fragments of logic systems underlying anticipatory reasoning or reasoning about actions. The paper also gave a requirement analysis for the development and maintenance environment for various ARRSs, and presented an architecture of the environment we are developing. By using our development and maintenance environment, developers and maintainers can develop and maintain various practical ARRSs more easily.

This work is ongoing work. There are many future works: to implement the common components, to modify an information security engineering environment (ISEE) as the development and maintenance environment, to implement the Theory Grid as a provider of fragments of logic systems.

References

[1] Cheng, J.: Anticipatory Reasoning-Reacting Systems Proc. International Conference on Systems, Development and Self-organization, (2002) 161– 165

[2] Cheng, J.: Temporal Relevant Logic as the Logical Basis of Anticipatory Reasoning-Reacting Systems. In Dubois, D. M., ed.: Computing Anticipatory Systems: CASYS - Sixth International Conference. AIP Conference Proceedings 718., The American Institute of Physics (2004) 362–375

[3] Cheng, J.: Connecting Components with Soft System Buses: A New Methodology for Design, Development, and Maintenance of Reconfigurable, Ubiquitous, and Persistent Reactive Systems. Proc. 19th International Conference on Advanced Information Networking and Applications, IEEE Computer Society Press (2005) 667–672

[4] Cheng, J.: Comparing Persistent Computing with Autonomic Computing. Proc. 11th International Conference on Parallel and Distributed Systems, IEEE Computer Society Press (2005) 428–432

[5] Cheng, J.: Persistent Computing Systems as Continuously Available, Reliable, and Secure Systems. Proc. 1st International Conference on Availability, Reliability and Security, IEEE Computer Society Press (2006) 631–638

[6] Cheng, J.: Temporal Deontic Relevant Logic as the Logical Basis for Decision Making Based on Anticipatory Reasoning. Proc. 2006 IEEE Annual International Conference on Systems, Man, and Cybernetics, The IEEE Systems, Man, and Cybernetics Society (2006) 1036–1041

[7] Cheng, J., Shang, F.: Persistent Computing Systems as An Infrastructure of Computing Anticipatory Systems. International Journal of Computing Anticipatory Systems **18** (2006) 61–74

[8] Cheng, J., Nara, S., Goto, Y., : FreeEnCal: A Forward Reasoning Engine with General-Purpose. In Apolloni, B., Howlett, R. J., Jain, L. C., eds.: Knowledge-Based Intelligent Information and Engineering Systems, 11th International Conference, KES 2007, XVII Italian Workshop on Neural Networks, Vietri sul Mare, Italy, September 12-14, 2007, Proceedings, Part II. Volume 4693 of Lecture Notes in Artificial Intelligence (Subseries of Lecture Notes in Computer Science)., Springer-Verlag (2007) 444–452

[9] Cheng, J., Goto, Y., Nara, S., Koh, T.: A Cooperative Grid Computing Approach to Automated Theorem Finding and Automated Problem Proposing. In Apolloni, B., Howlett, R. J., Jain, L. C., eds.: Knowledge-Based Intelligent Information and Engineering Systems, 11th International Conference, KES 2007, XVII Italian Workshop on Neural Networks, Vietri sul Mare, Italy, September 12-14, 2007, Proceedings, Part II. Volume 4693 of Lecture Notes in Artificial Intelligence (Subseries of Lecture Notes in Computer Science)., Springer-Verlag (2007) 840–851

[10] Cheng, J., Goto, Y., Morimoto, S., Horie, D.: A Security Engineering Environment Based on ISO/IEC Standards: Providing Standard, Formal, and Consistent Supports for Design, Development, Operation, and Maintenance of Secure Information Systems. Proc. 2nd International Conference on Information Security and Assurance, IEEE Computer Society Press (2008) 350–354

[11] Cheng, J.: Anticipatory Reasoning about Mobile Objects in Anticipatory Reasoning-Reacting Systems. In Dubois, D. M., ed.: Computing Anticipatory Systems: CASYS - Eighth International Conference. AIP Conference Proceedings 1051., The American Institute of Physics (2008) 244–254

[12] Cheng, J.: Adaptive Prediction by Anticipatory Reasoning Based on Temporal Relevant Logic. Proc. 8th International Conference on Hybrid Intelligent Systems, IEEE Computer Society Press (2008) 410–416

[13] Cheng, J., Goto, Y., Horie, D., Miura, J., Kasahara, T., Iqbal, A.: Development of ISEE: An Information Security Engineering Environment. Proc. 7th IEEE International Symposium on Parallel and Distributed Processing with Applications, IEEE Computer Society Press (2009) 505–510

[14] Dubois, D. M.: Computing Anticipatory Systems with Incursion and Hyperincursion. In Dubois, D. M., ed.: Computing Anticipatory Systems: CASYS - First International Conference. AIP Conference Proceedings 437., The American Institute of Physics (1998) 3–29

[15] Dubois, D. M.: Introduction to Computing Anticipatory Systems. International Journal of Computing Anticipatory Systems **2** (1998) 3–14

[16] Dubois, D. M.: Review of Incursive, Hyperincursive and Anticipatory Systems - Foundation of Anticipation in Electromagnetism. In Dubois, D. M., ed.: Computing Anticipatory Systems: CASYS'99 - Third International Conference. AIP Conference Proceedings 517, The American Institute of Physics (2000) 3–30

[17] Finkelstein, A., Kramer., J.: Software Engineering: a Roadmap. Proc. the Conference on The Future of Software Engineering, International Conference on Software Engineering, ACM (2000) 3–22

[18] Goto, Y., Nara, S., Cheng, J.: Efficient Anticipatory Reasoning for Anticipatory Systems with Requirements of High Reliability and High Security. International Journal of Computing Anticipatory Systems **14** (2004) 156–171

[19] Goto, Y., Endo, T., Cheng, J.: Continuous Reactability of Persistent Computing Systems. International Journal of Computing Anticipatory Systems **20** (2008) 219–229

[20] Harnel, D., Pnueli, A.: On The Development of Reactive Systems. In Apt, K. R., ed.: Logic and Models of Concurrent Systems. Springer-Verlag (1989) 477–498

[21] Harrison, W., Ossher, H., Tarr, P.: Software Engineering Tools and Environments: a Roadmap. Proc. the Conference on The Future of Software Engineering, International Conference on Software Engineering, ACM (2000) 261–277

[22] Kitajima, N., Nara, S., Goto, Y., Cheng, J.: Fast Qualitative Reasoning about Actions for Computing Anticipatory Systems. Proc. 3rd International Conference on Availability, Reliability and Security, IEEE Computer Society Press (2008) 171–178

[23] Kitajima, N., Nara, S., Goto, Y., Cheng, J.: A Deontic Relevant Logic Approach to Reasoning about Actions in Computing Anticipatory Systems. International Journal of Computing Anticipatory Systems **20** (2008) 177–190

[24] Kitajima, N., Goto, Y., Cheng, J.: Development of a Decision-Maker in an Anticipatory Reasoning-Reacting System for Terminal Radar Control. In Corchado, E., Wu, X., Oja, E., eds.: Hybrid Artificial Intelligence Systems, 4th International Conference, HAIS09, Salamanca, Spain, June 10-12, Proceedings. Volume 5572 of Lecture Notes in Artificial Intelligence (Subseries of Lecture Notes in Computer Science)., Springer-Verlag (2009) 68–76

[25] Nara, S., Shang, F., Omi, T., Goto, Y., Cheng, J.: An Anticipatory Reasoning Engine for Anticipatory Reasoning-Reacting Systems. International Journal of Computing Anticipatory Systems **18** (2006) 225–234

[26] Pnueli, A.: Specification and Development of Reactive Systems. In Kugler, H.J., ed.: Information Processing 86, North-Holland/IFIP (1986) 845–858

[27] Rosen, R.: Anticipatory Systems - Philosophical, Mathematical and Methodological Foundations. Pergamon Press (1985)

[28] Shang, F., Nara, S., Omi, T., Goto, Y., Cheng, J.: A Prototype Implementation of an Anticipatory Reasoning-Reacting System. In Dubois, D. M., ed.: Computing Anticipatory Systems: CASYS - Seventh International Conference. AIP Conference Proceedings 839., The American Institute of Physics (2006) 401–414

[29] Selim, M. R., Endo, T., Goto, Y., Cheng, J.: Distributed Hash Table Based Design of Soft System Buses. Proc. 2nd International Conference on Scalable Information Systems, ACM (2007) 1–4