A Transformation Mechanism between Sensory Data and Logical Formulas for Anticipatory Reasoning-Reacting Systems

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

Abstract
Anticipatory reasoning-reacting systems (ARRSs) were proposed as a new generation of reactive systems. Prediction and decision-making are important facilities of ARRSs. For the facilities, a prediction method and a decision-making method with forward reasoning based on strong relevant logic systems are proposed. On the other hand, practical reactive systems generally get sensory data and own internal status as character strings, but not as logical formulas. To implement facilities of prediction and decision-making based on the proposed methods, a transformation mechanism between observed data and logical formulas is demanded, but such a mechanism has not been proposed until now. This paper presents a transformation mechanism observed data and logical formulas for ARRSs. The mechanism can be applied to any computing anticipatory systems with logic-based reasoning.
Keywords : Anticipatory reasoning-reacting systems, Forward reasoning, Formal logic system, Data transformation.

1 Introduction

The concept of an anticipatory system first proposed by Rosen in 1980s [32]. Rosen considered that "an anticipatory system is one in which present change of state depends upon future circumstance, rather than merely on the present or past" and defined an anticipatory system as "a system containing a predictive model of itself and/or its environment, which allows it to change state at an instant in accord with the model's prediction to a latter instant." Dubois proposed the anticipatory system as a computing system, i.e., computing anticipatory system [13, 14].

On the other hand, from the viewpoints of software reliability engineering and information security engineering, what we need is really useful systems with anticipatorily predictive capability to take anticipation for forestalling disasters and attacks rather than the philosophical definition and intention of an anticipatory system. To develop anticipatory systems useful in the real world, Cheng proposed a new type of reactive systems, named "Anticipatory Reasoning-Reacting Systems," [1] as a certain class of computing anticipatory systems.

Anticipatory reasoning-reacting systems (ARRSs) were proposed as a new generation of reactive systems with high reliability and high security such that 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. In other words, an ARRS is a reactive system with facility of prediction and decision-making.

Prediction and decision-making are important facilities of ARRSs. For the facilities, a prediction method and a decision-making method with forward reasoning based on strong relevant logic systems are proposed [2, 3, 6, 7, 19, 20]. The proposed methods deal with already known empirical theorems and hypotheses, and observed facts represented as logical formulas. However, practical reactive systems generally get sensory data and own internal status as character strings, but not as logical formulas. To deal with observed sensory data and internal status in the proposed methods, it is necessary to interpret what fact the data shows and to transform the fact into a logical formula. To implement facilities of prediction and decision-making based on the proposed methods, a transformation mechanism between observed data and logical formulas is demanded, but such a mechanism has not been proposed until now.

Because many current reactive systems store observed sensory data and own internal status into relational database systems in the systems, if the transformation mechanism is compatible with relational database systems generally used, developers of ARRSs can implement ARRSs more easily.

This paper presents a transformation mechanism from observed sensory data and own internal status to logical formulas for ARRSs. The paper gives a requirement analysis for the transformation mechanism, proposes a design of a transformation mechanism with relational database management systems, and discusses its implementation issues. By using the proposed transformation mechanism, developers of ARRSs can modify existing reactive systems into ARRSs more easily. Moreover, the mechanism can be applied to any computing anticipatory systems with logic-based reasoning.

2 Anticipatory Reasoning-Reacting Systems

2.1 Logic-based Forward Reasoning on ARRS

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 issues: how to predict future event or events, and how to take next actions. For the facilities, a prediction method and a decision-making method with forward reasoning based on strong relevant logic systems are proposed [2, 3, 6, 7, 19, 20].

Reasoning is the process of drawing new conclusions from given premises, which are already known facts or previously assumed hypotheses (Note that how to define the notion of 'new' formally and satisfactorily is still a difficult open problem until now). In general, a reasoning consists of a number of arguments (or inferences) in some order. An argument is a set of statements (or declarative sentences) of which one statement is intended as the conclusion, and one or more statements, called 'premises,' are intended to provide some evidence for the conclusion. An argument is a conclusion standing in relation to its supporting evidence. In an argument, a claim is being made that there is some sort of evidential relation between its premises and its conclusion: the conclusion is supposed to follow from the premises, or equivalently, the premises are supposed to entail the conclusion. Therefore, the correctness of an argument is a matter of the connection between its premises and its conclusion, and concerns the strength of the relation between them (Note that the correctness of an argument depends neither on whether the premises are really true or not, nor on whether the conclusion is really true or not). Thus, there are some fundamental questions: What is the criterion by which one can decide whether the conclusion of an argument or a reasoning really does follow from its premises or not? Is there the only one criterion, or are there many criteria? If there are many criteria, what are the intrinsic differences between them? It is logic that deals with the validity of argument and reasoning in general.

A logically valid reasoning is a reasoning such that its arguments are justified based on some logical validity criterion provided by a logic system in order to obtain correct conclusions (Note that here the term 'correct' does not necessarily mean 'true'). Today, there are so many different logic systems motivated by various philosophical considerations. As a result, a reasoning may be valid on one logical validity criterion but invalid on another.

In general, a formal logic system L consists of a formal language, called the object language and denoted by $F(L)$, which is the set of all well-formed formulas of L, and a logical consequence relation, denoted by meta-linguistic symbol $\vdash_L$, such that $P \subseteq F(L)$ and $c \in F(L)$, $P \vdash_L c$ means that within the frame work of L, c is valid conclusion of premises P, i.e., c validly follows from P. For a formal logic system $(F(L), \vdash_L)$, a logical theorem t is a formula of L such that $\phi \vdash_L t$ where ϕ is empty set. We use $Th(L)$ to denote the set of all logical theorems of L. $Th(L)$ is completely determined by the logical consequence relation $\vdash_L$. According to the representation of the logical consequence relation of a logic, the logic can be represented as a Hilbert style formal system, a Gentzen natural deduction system, a Gentzen sequent calculus system, or other type of formal system.

Let $(F(L), \vdash_L)$ be a formal logic system and $P \subseteq F(L)$ be a non-empty set of sentences (i.e. closed well-formed formulas). A formal theory with premises P based on L, called a L-theory with premises P and denoted by $T_L(P)$, is defined as $T_L(P) =_{df} Th(L) \cup Th^e_L(P)$, and $Th^e_L(P) =_{df} \{et | P \vdash_L et \text{ and } et \notin Th(L)\}$ where $Th(L)$ and $Th^e_L(P)$ are called the logical part and the empirical part of the formal theory, respectively, and any element of $Th^e_L(P)$ is called an empirical theorem of the formal theory. Figure 1 shows the relationship among $F(L)$, $Th(L)$, $Th^e_L(P)$, and $T_L(P)$.

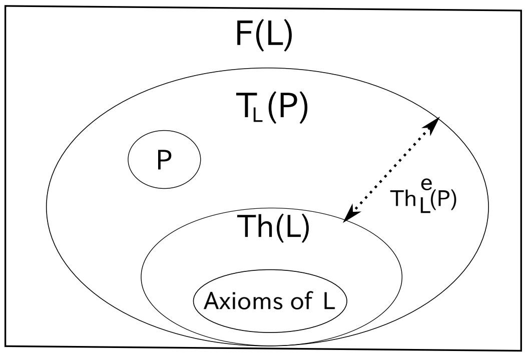


Fig. 1: *L*-theory with premises *P*

Automated reasoning is concerned with the execution of computer programs that assist in solving problems requiring reasoning. By adopting a suitable formal logic system for a target domain, we can do logically valid reasoning and get unknown or undecidable facts/hypotheses from empirical theorems that are well-known theories in a target domain. To do such logically valid reasoning automatically, a mechanism of automated reasoning is demanded. 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 facility to do reasoning automatically can be implemented by such forward reasoning engines and logic systems that are suitable for a target domain.

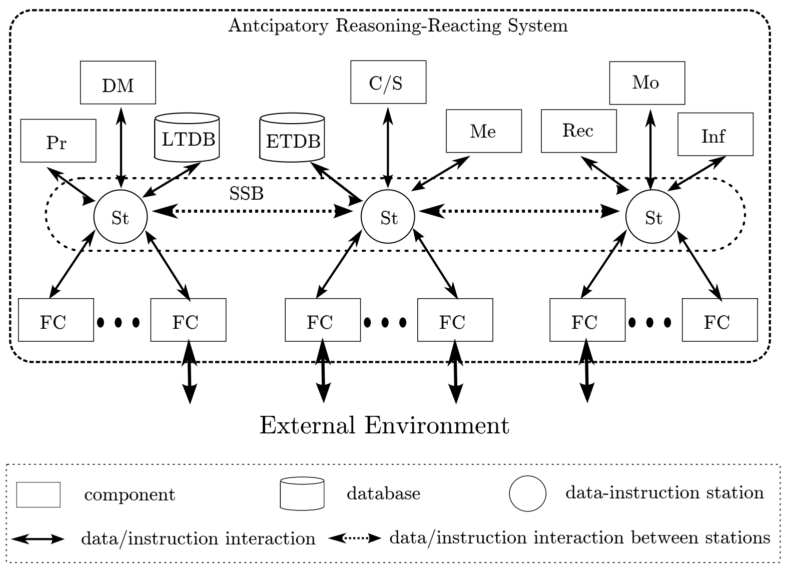
2.2 Overview of ARRS

A method using anticipatory reasoning based on temporal relevant logics or 3D spatio-temporal relevant logics was proposed [2, 6]. *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, 6]. Furthermore, to perform anticipatory reasoning automatically, an anticipatory reasoning engine was proposed and its prototype was implemented [7, 15, 25]. An anticipatory reasoning engine is a forward reasoning engine to perform anticipatory reasoning based on temporal relevant logics or 3D spatio-temporal relevant logics.

On the other hand, a decision-making method with reasoning about actions was proposed [19, 20, 21]. *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. The decision-making method uses reasoning about actions to enumerate candidates of next actions. *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 [20]. Deontic relevant logics and temporal deontic relevant logics are adopted as hopeful candidates of right fundamental logic systems for reasoning about actions [3, 19, 20]. Furthermore, to perform reasoning about actions automatically, an action reasoning engine was proposed and its prototype was implemented [19, 20]. Like the anticipatory reasoning engine, an action reasoning engine is a forward reasoning engine to perform reasoning about actions based on deontic relevant logics or temporal deontic relevant logics.

In anticipatory reasoning and reasoning about actions, logical theorems of a logic system are used as acceptable theories in any target domain while observed data, i.e., sensory data and internal status of an ARRS, theories in a certain target domain that the ARRS deal with, and predicted events or candidates of next actions are used as empirical theorems. Empirical theorems that represent theories in a target domain can be classified into three kinds. First one is a set of empirical theorems that represent static relationship among recognized objects in a target domain and static features of each object. An *object* in a target domain is an entity that can cause to change behavior of an ARRS directly. We named a set of explicitly known such empirical theorems a *world model*. Second one is a set of empirical theorems that represent mechanisms of occurrences of events and cause-and-effect relationships among occurrences of events. An *event* is a change of relationship among objects or a change of status of each object. We named a set of explicitly known such empirical theorems a *predictive model*. Third one is a set of empirical theorems that represent behaviors of objects, i.e., which object reacts to which event and how the object reacts. We named a set of explicitly known such empirical theorems a *behavior model*. According to figure1, in anticipatory reasoning, temporal relevant logics or 3D spatio-temporal relevant logics is used as a logic system L; observed data, a world model and a predictive model of a target domain are used as premises P; predicted events are included in $Th^e_L(P) - P$. Similarly, in reasoning about actions, deontic relevant logics and temporal deontic relevant logics is used as a logic system L; observed data, predicted events, a world model and a behavior model of a target domain are used as premises P; candidates of next actions are included in $Th^e_L(P) - P$.

An architecture of an ARRS was proposed [16]. Figure 2 shows the architecture of an ARRS. An ARRS is a persistent computing system [4] . A persistent computing system



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. 2: An architecture of an anticipatory reasoning-reacting system

can be constructed by a group of control components that are independent of systems, a group of functional components (FCs) that carry out special tasks of the system, and soft system buses (SSBs). Control components may include a central controller/scheduler (C/S), a central measurer (Me), a central recorder (Rec), a 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 such reports from a central monitor, and informs the reports to operators of the system. A soft system bus 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 of components; ones are common components in all ARRSs and others are application-dependent components. In figure2, an application-dependent component is represented as “FC.” The common components are a predictor (Pr), a decision-maker (DM), a logical theorem database (LTDB), and an empirical theorem database (ETDB). A predictor does anticipatory reasoning for deducing future events by using a forward reasoning engine, and chooses nontrivial predicted events according to selection rules given by developers of the ARRS. Then, it sends the predicted events to a decision-maker. The predictor takes observed data, a world model, and predictive model as input. The decision-maker does reasoning about actions for deducing candidates of next actions by using a forward reasoning engine, and chooses next actions from the candidate according to selection rules given by developers of the ARRS. Then, it sends next actions as instructions to application-dependent components related with the next actions. The decision-maker takes observed data, the predicted events sent from the predictor, a world model, and behavior model as input. A logical theorem database stores logic theorems of logic systems underlying anticipatory reasoning or reasoning about actions. An empirical theorem database stores observed data, empirical theorems of a world model, a predictive model, and a behavior model, and empirical theorems deduced by the predictor or the decision-maker.

PCS-core components are control components and soft system buses. They are common in all persistent computing systems. ARRS-core components are a predictor, a decision-maker, a logical theorem database, and an empirical theorem database. They are common components in all ARRSs, but not in all persistent computing systems. One of our ultimate goals is to provide PCS-core and ARRS-core components as a development framework of ARRSs to their developers.

3 Requirement Analysis for transformation mechanisms

To implement facilities of prediction and decision-making based by using forward reasoning engine, any ARRS should have a mechanism to transform sensory data and logical formulas. FreeEnCal [5] 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. It can interpret specifications written in the formal language such that any user can use the formal language to describe and represent formulas and inference rules for deductive, inductive, and abductive reasoning. It also can reason out all or a part of logical theorem schemata of a logic system under the control conditions attached to the reasoning task specified by users, and all or a part of empirical theorems of a formal theory and facts under the control conditions attached to the reasoning task specified by users. FreeEnCal can deal with only logical formulas. However, practical reactive systems generally get sensory data and own internal status as character strings. To implement facilities of prediction and decision-making based by using forward reasoning engine like FreeEnCal, a transformation mechanism between observed data and logical formulas is demanded.

The requirements the mechanism should satisfy are as follows.

R1: *The transformation mechanism should be event-driven.* An ARRS is a kind of reactive systems. ARRSs should react to sensory data that come from its outside environment. In addition, ARRSs should react to changes of its internal status, too. To react to the sensory data or the changes, ARRSs should do prediction and decision-making. By the way, in any information system, such the sensory data or the changes are represented as character strings. Therefore, it is necessary to transform the character strings into logic formulas when sensory data and/or changes of internal status occur.

R2: *The transformation mechanism should be compatible with relational database management systems.* Many current reactive systems use relational database management systems, such as IBM DB2 [18], Oracle database [27], MySQL [26], PostgreSQL [30], etc, to construct a database managing sensory data and own internal status in the systems. To implement ARRSs more easily, the transformation mechanism should be compatible with such relational database management systems.

R3: *The transformation mechanism should generate logical formulas according to transformation rules given by developers of an ARRS.* Kinds of sensory data or internal status observed in a system are different from each ARRS. Although kinds of observed data are same, how to use the observed data may be different from each ARRS. Kinds of logical formulas and how to make them are different if interpretation of observed data are different. Moreover, only developers of each ARRS can know those things. Hence, the transformation mechanism should provide environment to input transformation rules which are sets of procedures to generate logical formulas from observed data, and it should be able to do the transformation according to the rules.

R4 *The transformation mechanism should revise already deduced empirical theorems.* When a new logical formula is generated from new observed data, the logical formula may conflict with an already deduced/generated empirical theorem. In that case, it is necessary to reduce the empirical theorem which conflicts with the new logical formula and other empirical theorems that deduced from the empirical theorem.

R5 *The transformation mechanism should be implemented as ARRS-core components.* Because facilities of prediction and decision-making with forward reasoning engine are common facilities among all of ARRSs, any ARRS should have such mechanism.

4 A Transformation Mechanism

Figure 3 shows a design of a transformation mechanism. The transformation mechanism does transformation from sensory data or changes of its internal status to logical formulas when instructions to create, update, and delete data are sent to a database that manages already observed data in an ARRS. Before running an ARRS, its developers or operators store transformation rules into the mechanism. After running the ARRS, when the ARRS gets some sensory data or changes of internal status, it stores those observed data into the relational database (RDB), i.e., ‘create data’, replaces already stored observed data with the new one, i.e., ‘update data’, or deletes already stored data that are inconsistent with the new one, i.e., ‘delete data’. Note that RDB is constructed on a relational database management systems. At that point of time, a filter takes the instruction for RDB from outside of the translation mechanism. And then, it sends the instructions to RDB, and calls a generator. The generator does transformation according to the given transformation rules.

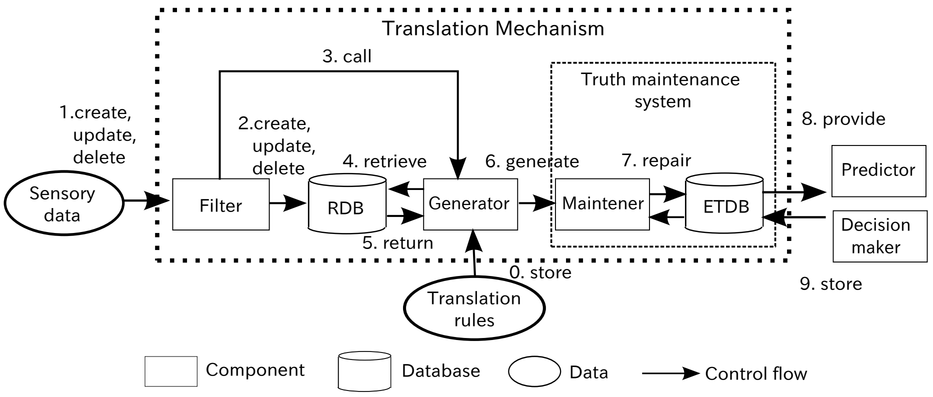


Fig. 3: Control flow of a transformation mechanism

It retrieves observed data already stored in RDB, then it judges whether retrieval results can satisfy each transformation rule or not. If satisfy, the generator generates logical formulas according the transformation rule. The generated logical formulas are sent to a maintainer. The maintainer checks whether there are contradictional formulas against the generated logical formulas in a empirical theorem database (ETDB) where ETDB is explained at section 2.2. When the contradictional formulas exist, the maintainer tries to keep consistency of set of empirical theorems in ETDB. If the contradictional formulas are already stored observed data, the maintainer replaces contradictional formulas and delete all empirical theorems deduced from the contradictional formulas autonomously. If the contradictional formulas are empirical theorems belonging to a world, a predictive, or a behavior model, the maintainer enumerates all empirical theorems deduced from the contradictional formulas, and informs operators of the ARRS the occurrence of contradiction and enumerated empirical theorems. If the contradictional formulas are empirical theorems deduced from already stored observed data and/or empirical theorems belonging to the models, the maintainer enumerates all empirical theorems deduced from the contradictional formulas and all formulas that occur in derived paths of the contradictional formulas, and informs operators of the ARRS the occurrence of contradiction and enumerated empirical theorems. The reason why the maintainer should change its operation depending on kinds of contradictional formulas is that how to repair inconsistency of a set of managed empirical theorems in ETDB depends on purpose of each ARRS. ETDB provides empirical theorems if the predictor or the decision-maker requires them. ETDB also gets empirical theorems deduced by the predictor or the decision-maker.

The mechanism satisfies all of requirements defined in section 3. To satisfy R1, the mechanism has a filter. Events that triggers prediction and decision-making are to create, update, and delete data on a database in an ARRS so that the mechanism makes the filter monitor the instruction for the database. To satisfy R2 and R3, the mechanism uses relational database management system to construct a database that stores observed data, and facility to generate logical formulas is designed as a generator, which is an independent component from the relational database management system. To satisfy R4, the mechanism has a maintainer. R5 is satisfied because all of component in the mechanism are independent from a target domain of any ARRS. Moreover, the mechanism is independent from FreeEnCal so that it can be used with any logic-based reasoning engine. We will discuss how to implement the generator and maintainer in next section.

The proposed mechanism is a kind of active database systems. Active database systems are systems that can respond automatically to events that are taking place either inside or outside the database system itself [29]. To realized the reactive behavior, most active database systems use rules that have up to three components; an event, a condition, and an action. The *event* part of a rule describes a happening to which the rule may be able to respond. The *condition* part of the rule examines the context in which the event has taken place. The *action* describes the task to be carried out by the rule if the relevant event has taken place and the condition has evaluated to true. Such a rule with three components is known as an event-condition-action or ECA-rule [29]. Coming the instructions to a database in an ARRS can be regarded as events. Generating logical formulas according to transformation rules can be regards as reactive behavior to the events. Transformation rules can be regarded as ECA-rules. Hence, the transformation mechanism is an active database system for ARRSs.

The proposed mechanism is not a deductive database system. Deductive database systems are database management systems whose query language and (usually) storage structure are designed around a logical model of data [31]. In deductive database systems, we can use most of facilities that logic programming languages like Prolog provide. The purpose of proposed mechanism is to generate logical formulas. Deductive database systems can list up atomic logical formulas (first order predicate) that satisfied rules (conditions) given by its users, but cannot generate logical formulas except Horn-clause style. The proposed mechanism has to generate logical formulas represented as not only Horn-clause style, but also other forms. On the other hand, most of deductive database systems are implemented as one database management system, but not a system that wraps existing relational database systems. We do not adopt already existing relational database management systems to construct a database in an ARRS if we adopt deductive database system as a part of transformation mechanism. Moreover, in ARRSs, the facility of reasoning is provided by forward reasoning engine in a predictor and a decision-maker. Therefore, facilities which logic programming languages provide are over-spec for the transformation mechanism.

5 Implementation Issues

To implement such proposed mechanism, there are implementation issues as follows. How does the generator deal with many kinds of relational database management systems? How does the generator provide an environment that developers or operators of an ARRS can describe transformation rules easily? How does the maintainer keep consistency of empirical theorems in ETDB?

The generator should deal with many kinds of relational database management systems because of satisfying R2 in section 3. One of difficulties of dealing with many kinds of relational database management systems is dialects of SQL among them. Schemata of a database that stores observed data in an ARRS may be different from other ARRS. Because only developers of the ARRS can know the schemata, they should describe queries to the database in transformation rules. For developers of ARRSs, representation of transformation rules should be unified while they use any kinds of relational database management systems. If SQL is used for description of transformation rules, it is difficult to satisfy the above requirement. By the way, object-relational mapping is a mechanism that conversion of data held in objects to a form that can be stored in a relational database and vice versa [28]. By using object-relational mapping, developers of ARRSs can write transformation rules by using object-oriented programming language without considering the differences among dialects of SQL.

The generator should provide an environment that developers of an ARRS can describe transformation rules easily. If developers of ARRSs use an object-oriented programming language with object-relational mapping mechanism to describe the transformation rules, we can consider that the generator provides an environment to describe the transformation rules in unified way. However, to describe transformation rules, object-oriented programming languages are too much of expressive power. Such expressive power causes software bugs and/or vulnerabilities. Moreover, it is difficult or cost consuming for the developers to program procedures of generating logical formulas. A transformation rule can be regarded as a constraint to find records, which satisfy conditions given by developers of ARRSs, from tables or views of a relational database. Under the consideration, constraint logic programming can be used for a technique to describe transformation rules easily. Constraint logic programming languages are logic programming languages in which unification is replaced by constraint solving in various domains [8]. Constraints are special predicates whose satisfiability can be established for various domains using efficient algorithms (e.g., inequalities and disequalities). Unification can be viewed as a particular type of constraint that tests equality in the domain of trees. If a transformation rule consists of only constraints without procedures to generate logical formulas, cost to describe the rule becomes low.

The maintainer should keep consistency of empirical theorems in ETDB. The system that provides a facility to keep consistency of set of logical formulas is called as a truth maintenance system [10]. We can expect that results of studies for truth maintenance systems (TMSs) are able to be used for implementing the maintainer. However, we cannot adopt traditional TMSs as mechanism of the maintainer directly. Some traditional TMSs [11, 12, 22, 23, 24, 33] cannot deal with family of strong relevant logics, i.e., temporal relevant logics, 3D spatio-temporal relevant logics, deontic relevant logics, and temporal deontic relevant logics, because those TMSs require a certain logic system underlying the mechanism to keep consistency of managed logical formulas, e.g., classical mathematical logic or its conservative extensions, or logic systems dealing with uncertainty. Other traditional TMSs [9, 10] can deal with the family of strong relevant logics, but the TMSs are not optimized for dealing with them. Therefore, a TMS dealing with the family of strong relevant logic is demanded. We have proposed and been developing such TMS [17].

6 Concluding Remarks

This paper has presented a requirement analysis of transformation mechanisms from sensory data to logical formulas, shown a design of the mechanism, and investigated implementation issues of the proposed mechanism. The mechanism is independent from a target domain of any ARRS, and the forward reasoning engine we are developing, so the mechanism can be applied to any computing anticipatory systems with logic-based reasoning.

Some future works are implementing a prototype of the proposed mechanism and verifying usefulness of the mechanism by some case studies.

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.: 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

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

[5] 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. Springer-Verlag (2007) 444–452

[6] Cheng, J., Goto, Y., Kitajima, N.: 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

[7] 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

[8] Cohen, J.: Logic Programming and Constraint Logic Programming. ACM Computing Surveys **28:1** (1996) 257–259

[9] de Kleer, J.: An Assumption-based TMS. Artificial Intelligence **28:2** (1986) 127–162

[10] Doyle, J.: A Truth Maintenance System. Artificial Intelligence **12:3** (1979) 231–272

[11] Dubois, D., Lang, J., Prade, H.: Handling Uncertain Knowledge in an ATMS Using Possibilistic Logic. Proc. ECAI Workshop Truth Maintenance System (1990) 87–106

[12] Dubois, D., Berre, D. L., Prade, H., Sabbadin, R.: Using Possibilistic Logic for Modeling Qualitative Decision: ATMS-based Algorithms. Fundamenta Informaticae **37:1-2** (1999) 1–30

[13] 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

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

[15] 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

[16] Goto, Y., Kuboniwa, R., Cheng, J.: Development and Maintenance Environment for Anticipatory Reasoning-Reacting Systems. International Journal of Computing Anticipatory Systems **24** (2008) 61–72

[17] Goto, Y., Cheng, J.: A Truth Maintenance System for Epistemic Programming Environment. Proc. 8th International Conference on Semantics, Knowledge and Grid. IEEE Computer Society Press (2012) 1–8

[18] IBM: DB2 Database Software. `http://www-01.ibm.com/software/data/db2/`, accessed at (2012)

[19] 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

[20] 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

[21] 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. Springer-Verlag (2009) 68–76

[22] McAllester, D. A.: An Outlook on Truth Maintenance. AI Memos **551** (1980)

[23] McDermott, D.: A General Framework for Reason Maintenance. Artificial Intelligence **50:3** (1991) 289–329

[24] Monai, F. F., Chehire, T.: Possibilistic Assumption based Truth Maintenance System, Validation in a Data Fusion Application. Proc. 8th international conference on Uncertainty in artificial intelligence. Morgan Kaufmann Publishers Inc (1992) 83–91

[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] Oracle: MySQL. `http://www.mysql.com/`, accessed at (2012)

[27] Oracle: Oracle Database. `http://www.oracle.com/us/products/database/overview/index.html`, accessed at (2012)

[28] Oxford University Press: A Dictionary of Computing, Sixth Edition. Oxford University Press (2008)

[29] Paton, N. W, Diaz, O.: Active Database Systems. ACM Computing Surveys **31:1** (1999) 63–103

[30] PostgreSQL Global Development Group: PostgreSQL. `http://www.postgresql.org/`, accessed at (2012)

[31] Ramakrishnan, R., Ullman, J. D.: A Survey of Deductive Database Systems. The Journal of Logic Programming **23:2** (1995) 125–149

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

[33] Shen, Q., Zhao, R.: A Credibilistic Approach to Assumption-Based Truth Maintenance. IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans **41:1** (2011) 85–96