Paper 4

by Paper Aradea

Submission date: 08-Oct-2020 03:46PM (UTC+0700)
Submission ID: 1408912022
File name: A4._Paper_IoTBookChapter.pdf (2.28M)
Word count: 10167
Character count: 54226

13

Self-Adaptive Cyber-City System

Iping Supriana, Kridanto Surendro, Aradea Dipaloka, and Edvin Ramadhan

CONTENTS

13.1 Introduction: Cyber-City Systems and Seven Pillars of Life

The growth of urban population in the world nowadays encourages the growth of the level of needs which is increasingly out of control. This is based on the fact that the level of knowledge in the population continues to increase, so (ITU-T, 2014) it led to the social, economic, and environmental issues becoming connected to each other. This fact shows that the characteristics of the system to control a city has the diversity of interrelated elements and will lead to a variety of challenges and problems in the provision of public services (PSn).

Therefore, the city management needs to have an intelligent solution and creates a sustainable environment to (ITU-T, 2014) manage various infrastructure resources, environmental resource, monitoring the activities in the city, and more needs. All of this management will relate to the system requirements of the city-management system, such as transportation management, energy management, water management, waste management, municipal administration management, health services management, and other PSn management.

Internet of Things (IoT) is a concept that can answer the challenges. Nowadays, there are several definitions of it, but basically, they have the same goal to extend the benefits of Internet connectivity that can connect a variety of real-world objects as physical and virtual representation continuously. One of the implementation of IoT for the needs of city-management system is how to utilizing the functions of information and communication technology is developed on a concept called the cyber city or smart city or intelligent city or some other term, that basically aimed to meet the requirement of city management development through the systematic formulation. The proposed system is packed with various facilities, such as the efficiency and effectiveness of implementation, speed of access, accuracy of services, and others. But this various advantages will not be able to make a significant contribution if only temporary. This is related to the detail and completeness of the concept to meet the characteristic of the growth in a city.

In this discussion, we will describe a model about how to establish a system that can accommodate a variety of changes in the application environment, how to manage the change, and how to make the system has the capability to adapt and fit the changes. So that this model can be an inspiration in developing a scenario of systems that needed to manage a city. The model that has been designed inspired by the seven pillars of life (taklif) (Nabulsi, 2010), which is composed of the universe, law, reason, nature, lust, freedom of choice, and time. These seven pillars are represented through an agent-oriented approach BDI (belief, desire, intention). So the points of the inspiration are, we need to consider which part is associated with the domain model and which part is associated with control model in building the management software system, in this case, the city-management system. It is expected to create a computational model that can be used as guidance in developing a system of cyber-city systems and other IoT software systems in general, with the self-adaptive ability which can handle the issue of change and growth of systems.

13.2 The Basis of System Development

Self-adaptive systems (SAS) is presented as an alternative solution to the problems associated with the complexity of the IoT system, including the demands of autonomy, automation, adaptability, flexibility, scalability, reliability, speed, and others (Supriana and Aradea, 2015). SAS is a system that can automatically take appropriate action based on the knowledge that the system has about what is happening in the system itself, guided by the goal, and assisted by the users who are given the access (Ganek and Corbi, 2015). SAS can modify the system behavior in response to the changes in the system itself or the changes in its environment (Cheng et al., 2014). This definition shows that the system has the knowledge, which can be used to achieve its objectives through several means, and is able to make the adaptation in behavior based on events in the environment.

The characteristics of each element in the environment of an IoT-management system require a mapping and alignment with the system that will be the ultimate driving machine in the achievement of a city manager. So (Aradea et al., 2014), all of the activity that occurs in the environment of this IoT system requires a mechanism that can represent the behavior of the system in real-world change. SAS concept was developed as a solution to this problem. The constructed cyber-city system must have the capability to reflect the requirements of each element of a city, to the adaptability of changes in the system and its environment.

13.2.1 Representation of System

Basically, the universe was created for man to be given the mandate as the leader on earth, who is able to organize and manage all the resources of the earth. This practice is called the taklif (Nabulsi, 2010). The process of taklif comprises seven pillars (Nabulsi, 2010), namely (ITU-T, 2014) (1) universe, (2) reasonable, (3) nature, (4) religion or laws, (5) lust, (6) the freedom to choose, and (7) time. The explanation of all seven pillars of this taklif as can be seen in Table 13.1.

TABLE 13.1

Seven Pillars of the Taklif System

Pillars	Explanation
The Universe	All content of the earth, including the behavior of the components of the universe, can be grouped into two kinds of roles in the taklif system, that is, as the knowledge and as utilization. Human reasoning works on the principles of harmony with the universe and the principles of law. There are three principles of human reasoning, namely the principle of *cause and effect* (something happens for a reason), the principle of *goal* (something exists must be the goal), and the principle of *anticontradictory* (two things can apply when they are opposites).
Reasonable	The human mind will process two facts, namely the fact sensuous (reality) and logical facts (explanation). The logical fact has a higher position than the sensuous fact. Humans have the potential to think broadly but limited by the information provided by the senses. There is a lot of information about the past and the future that cannot be reached by the senses. If we relate this thinking to the concept of religion, we can state that religion is the only thing that can be delimiters in the laws or reason.
Nature	Naturally, humans were created with moral instincts. This moral instinct that defines the rules of human life. But there are some human attitudes which are contrary to this rule, this attitude appears because of character flaws and physical to the human. This weakness makes the natural rule to be violated, distorted, and even erased. This weakness is a challenge for humans, this rule can be restored to its natural state through learning or management that is able to organize and disenchant the people to respect the rule of nature.
Religion or Laws	Religion is a concept of how people should behave toward the creator, fellow human beings, and its natural surroundings. The truth of human thought and the natural order must be calibrated with the rules of religion; truth in religion is generally considered an absolute, whereas reason and nature can be distorted by lust. To determine the truth as the basis of the rule, the truth must meet four requirements, which is carried by religion or laws, approved by human reasoning, in accordance with nature, and confirmed by objective reality.
Lust	Lust is the desire to achieve the goal, with their desire for a need for pleasure and love of something inherent in man, the lust will make him it will strive to achieve it. But lust is neutral. Lust is controlled by the choice of reason and nature, whether just as satisfying a need or as a means of achievement of objectives as stated in the rules of religion or laws.
Freedom of Choice	Freedom of choice is central to the taklif system. In this case, the religious rules known as the belief which acts as the basis of the freedom to choose. In this world, there are many choices and each choice has its own consequences. Religious rules and nature that will be a reference for these choices.
Time	All interactions of the six pillars of the taklif system which have been described previously run in the dimension of space and time. Its space is the earth and the universe, whereas the time is a series of events recorded since humans are born until his death.

Source: Nabulsi, R. M. *7 Pilar kehidupan: Alam semesta, akal, fitrah, syariat, syahwat, kebebasan memilih, dan waktu*, Gema Insani, Jakarta, 2010; Agustin, R. D., Supriana, I., Model Komputasi Pada Manusia dengan Pendekatan Agent, Kolaborasi BDI Model dan Tujuh Pilar Kehidupan sebagai Inspirasi untuk Mengembangkan Enacted Serious Game, *Konferensi Nasional Sistem Informasi* (*KNSI*), ITB, 2012.

13.2.2 Construction of Model

Based on the model of the representation of system in Section 13.2.1, the construction model of developed software in the IoT concept, referring to the seven pillars of the taklif system, is mapped to the BDI Model (Bratman, 1987; Mora, 1999; Russel, 2003). A more detailed description can be found in the paper (Agustin and Supriana, 2012). The BDI Model is a theory of reasoning from a practical standpoint (practical reasoning) and were adopted in an agent-oriented architecture.

There are two main processes in practical reasoning, namely deliberate and means-end reasoning. Deliberate is the process of deciding specific circumstances to be achieved, called the intention. The means-end reasoning is the process to draw up a plan to achieve that intention. The deliberate process begins with understanding the stimulus from the environment using the belief, that the perception of the agent (which is assumed to be true) on the environment. The outcome of this process is the captured problem classification by the agent at the detected time. Based on these results, the system will search for the best reaction to solve the given problem. In the desires of the BDI model, these solutions are emerging as an alternative option which generated sporadically, so the various alternative solutions appear; but because it is generated sporadically, there may be a solution that is unrealistic or inconsistent.

Before adding a goal to the state of intention, all alternative choices in the state for this goal needs to be verified and validated, to be realistic with the condition of the agent, and consistent with the objectives set (Mora, 1999). Furthermore, in the state of intention, the system has obtained some options, to be realized as an action. In the end, the mental state will generate an action into some subactions to be executed by the effector. The mapping of these BDI models of the seven pillars of the taklif system is shown in Figure 13.1.

1. Pillars of components in the universe are the facts of each entity and its environment. The behavior of the components in the universe in the form of events can be captured by the information-model context. In this case, there are two kinds of roles, namely the introduction of knowledge and utilization, which means how to recognize and utilize the facts.
2. The next component is the representation of the pillars of sense. Pillars of sense mean to know the nature and to work based on the principles that correspond to the goal, nature, and religion or law (rules of how to behave/delimiters sense). This component evaluates the condition by processing the facts of reality and logical facts based on occurred events. In this evaluation phase, the facts will be processed, diagnosed, and classified based on the problem, so we can determine how to choose understanding.
3. The processing of facts would trigger some response that will be taken by the action component in the process of adaptation. There will be some choices in response to the prevailing condition in accordance with the rule of lust (functional requirements and nonfunctional requirements), and to generate the response options, it will be processed by the system itself without being given the freedom to choose.
4. Components of sense will choose several response options in accordance with the model of rules in religion or law (as described). In this process, selection and filtering are performed between some alternatives of priority response, so that the results are just a few alternatives with the highest priority.

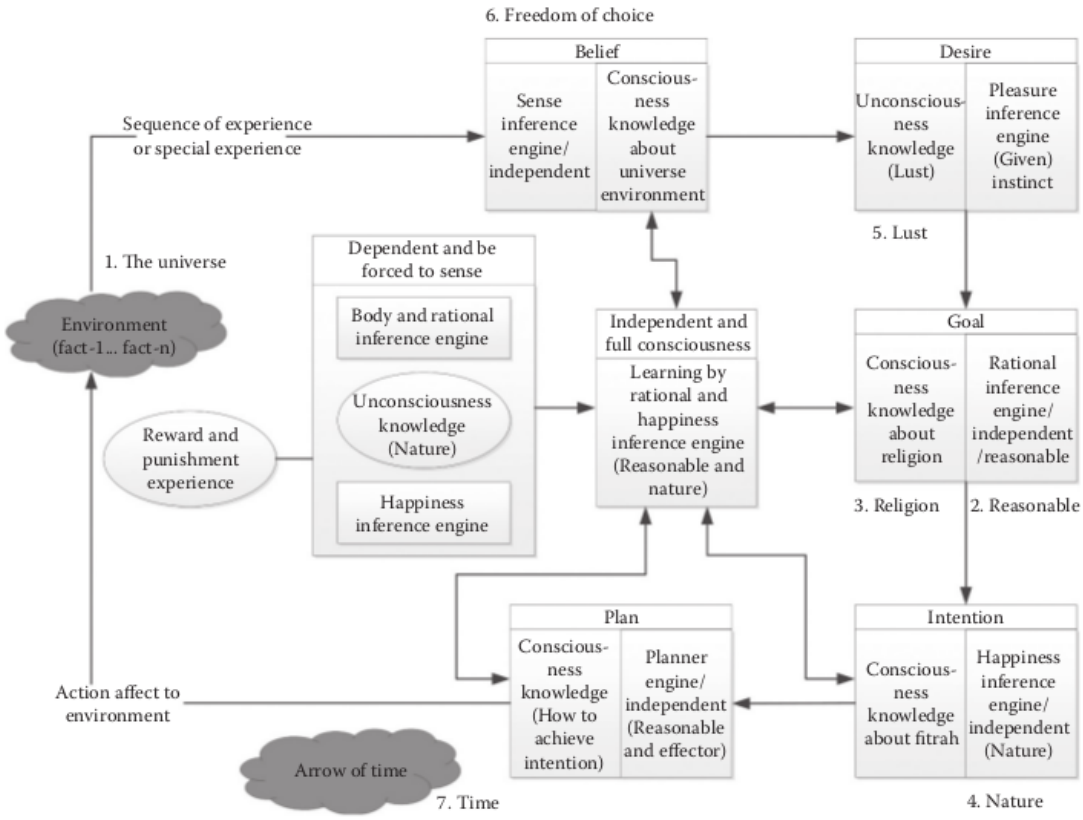

FIGURE 13.1
The mapping of the seven pillars in the taklif system into the BDI model.

5. Representation of the qolbu components will filter out some of the alternate options again from remaining responses, in reference to the understanding of nature's rule (instinctive understanding of the religion or the law of nature) until the obtained result is the best response that will be implemented.
6. Based on the selected response, some planning for the adjustment process can be arranged, and the effector of the body part system to execute the selected response can be determined in a certain order. The result of the execution of the effector is called as action.
7. The actions taken will get a reward or punishment based on the accuracy of the selected response with religious rules and nature should be. This stage will be a learning process for the system and will become a trigger for updating knowledge so that the system can determine the best response action in the future.

The characteristics of the seven pillars of the taklif system in conducting adaptation response are shown in Figure 13.1. The pillars are represented by 4 major rules in 3 BDI states, which is *desires* (R1), *goal* (R2 and R3), and the *intention* (R4); the states will

determine the understanding of *belief*. The development strategy for each of these rules are as follows:

1. *Rule-1* (*generic structures*): Represent the real world in the form of abstract classes to model the goal.
2. *Rule 2* (*conceptual rule*): Identifying the generic structure that meets the basic specifications of fact; this rule will also guide the system in determining an alternative initial option based on the understanding of belief to environmental stimuli.
3. *Rule-3* (*configuration rule*): Define the configuration of the components, verify and validate an alternative option of conceptual results of operation rule.
4. *Rule 4* (*specific rule*): A special rule that is used if needed or if operating results do not meet certain criteria to obtain one or more options that are most relevant actions.

13.3 Self-Adaptive Model

This section will discuss the proposed model to build software systems with self-adaptive capabilities in the IoT concept. The framework was developed as shown in Figure 13.2, consisting of two main parts, as follows:

1. The domain model is represented as goal model. The domain model is a component that provides the IoT concept's basic functions and application logic. This model was developed based on goal-oriented approach, which can exploit the human-oriented abstractions such as agents and some other concepts, so it can represent real-world conditions.
2. The control model is represented as inference engine that controls or manages the target system through the adaptation logic. These models apply some patterns of action of the agent through the transition rules. The control model considers context-aware scenarios to represent a control strategy to meet the requirements of self-adaptation.

The domain model is a requirement of the IoT-goal system. Domain models are mapped to the BDI models for defining goals, planning, and other elements. The results of the mapping are represented by agent definition file and the plan specification. The concept of this agent represents the context of the IoT system as a fact in belief base and environment class for the purposes of monitoring functions (Supriana and Aradea, 2016). This approach quite provides the variability at run-time, but we extend it through the mapping into the BDI models, which is based on seven-pillar taklif with rule-based, which are constructed in a more general and more flexible, so it can provide a better way of analyzing the variability and can help in detailing the behavior of the system to meet the goals and adaptation action.

The control model is the setting of the behavior of the IoT system, tasked to monitor the environment and adjust the system if necessary, for example, reconfigure when there is a change goal, optimize themselves when the operation changes, and able

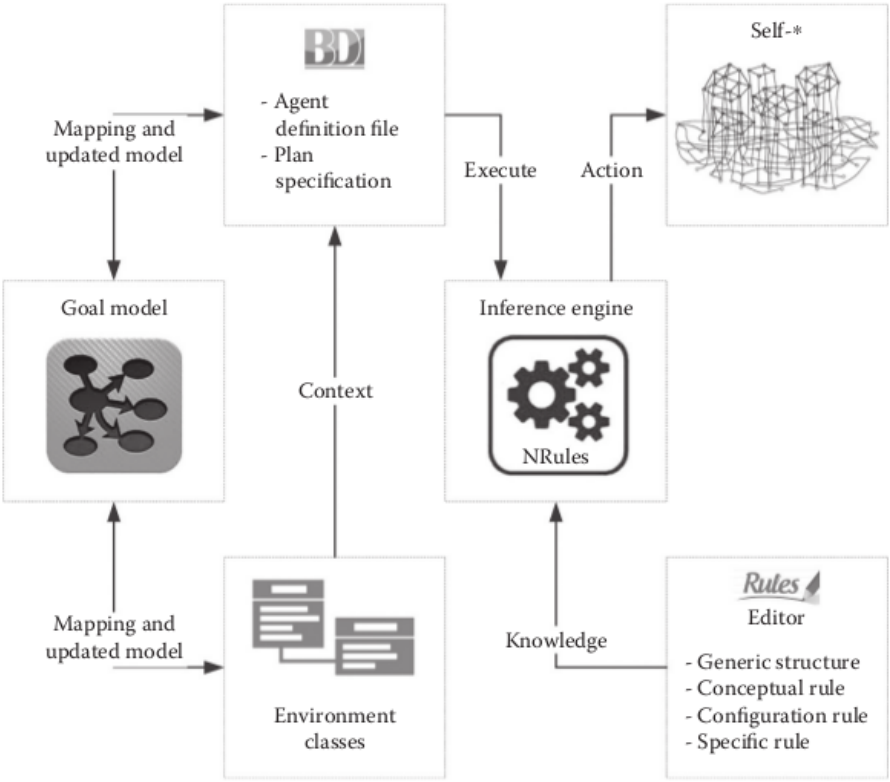

FIGURE 13.2
Framework for self-adaptive systems.

to handle certain types of errors. The pattern applied is an extension of the architecture event-condition-action (ECA) which in one or more *event* refers to the state of the current system. The models will change the application or IoT-system environment. Meanwhile, one or more *condition* refers to the time when a particular event occurs and the action rule is activated. So one or more *action* can be activated under certain conditions through the operation of rules in determining the behavior of the adaptive system.

13.3.1 Rule Representation

Any changes occurred in the IoT system is seen as a model of automata and the relationship of the possible state. Those changes can occur from an initial state until it reaches the final state. So the description of automata can be expressed as the model of rules which describes the relationship between the two states.

The representation of rules of these designed automata models, automatically, can detect the changes in the states and determine the necessary actions for adaptations. Scenarios are developed through the formation of the rule to identify similarities in the properties of each list of knowledge. In general, the rule-based system consists of 11 classes (Wu, 2004) as follows: rule, left-hand side (LHS), expression, fact, slot, composite expression, unary expression, binary expression, right-hand side (RHS), action, and function, as shown in Figure 13.3.

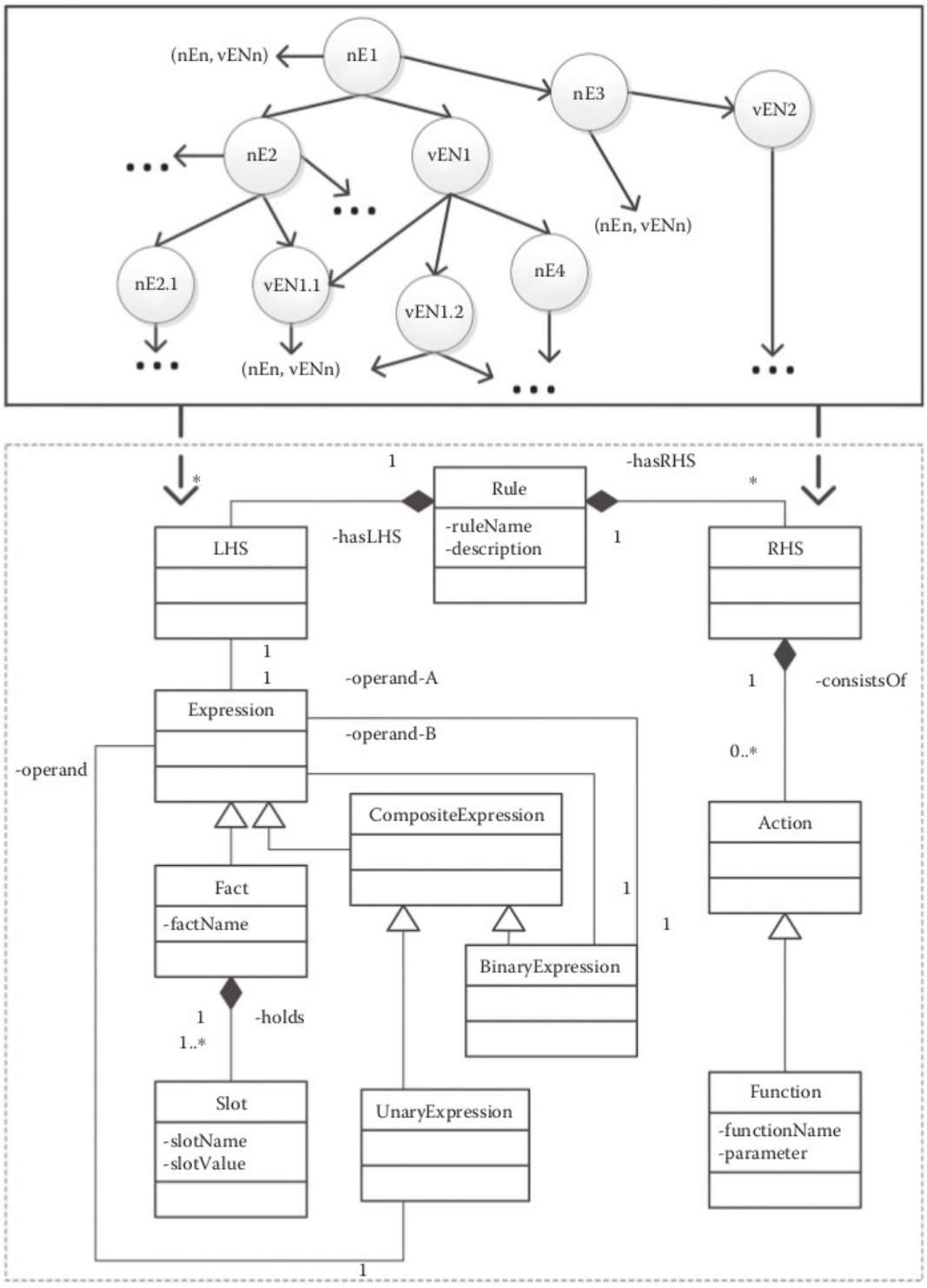

FIGURE 13.3
Knowledge tree and rule-based systems.

This rule-based system has a production rule in the form *if condition* > *then action* >. Action > on the RHS is concrete. This production rule is a practical consequence of a particular condition, whereas the other rules in the rule base are known as derivation rule. Conclusion > on the RHS is more abstract and becomes the logical consequence of certain conditions. However, a production rule can apply derivation rule using certain actions such as *assert* that expressed knowledge.

An action > defined on the RHS is determined by the expression on the LHS, which is conformity between rules and facts. Therefore, to create a self-adaptive ability, the rule-based system should be expanded as discussed in Section 13.3.2. The main objective is to equate the LHS in accordance with the formulation of the property slot, so there is a set of inner properties that become the source of facts, such as variables, constants, tuples, and so on. The basis of the mechanism is comparing the two or more collections of properties with a certain structure based on the criteria required.

In Figure 13.3 (bottom), we can see that the LHS is an expression that can be a fact, a single expression (pattern) which is characterized by a name and a collection of slots, or may be a composite expression with conditional elements (and, or, not). The conditional element is also used to connect a single expression (facts) or a composite expression. Class of fact has a containment reference in the class slot with the attribute "slotName" and "slotValue." The changes in this slot attribute will be detected based on any of the elements contained in the facts. For example, in the structure of the following basic rule:

$$\text{if } \langle (\text{fact-1})\ (\text{fact-2}) \ldots (\text{fact-}N) \rangle \text{ then } \langle (\text{action-1}) \ldots (\text{action-}N) \rangle \tag{13.1}$$

Information models of facts have variations on slotName and slotValue based on the defined information context. The information models are defined to be a relation between entities in the system and the environment. Information models are also represented as an interconnected class and the facts in working memory. Thus, a system entity would have a slotName such as "nameEntity" (nE1, nE2, nE3 … nE*n*) and has a view component for the environment as slot "viewEnvironment" (vEN) and can be composed of other facts, for example (vEN1, vEN2, vEN3,… vEN*n*,), and each slot is possible to have more specific (vEN1.1, vEN1.2, … vEN1.*n*), and the specification is also very possible to have other new slot. Representation of fact is a list of knowledge owned by each entity, and we define the list of this knowledge in a graph form or tree structure of knowledge, as we can see in Figure 13.3.

The tree structure of knowledge will add, delete, update, and customize every vertex automatically in response to the occurred state changes in accordance with the facts. The structure of property description of the knowledge tree can be managed by mapping the patterns of the seven pillars of taklif into the BDI model. So the representation of rules will be a reference for developed adaptation mechanisms to make a rule expansion of the ECA model (Daniele, 2006; Pires et al., 2008). This model can handle a wider scope of the solution space, and flexible, by using the following basic structure:

$$\begin{gathered}\text{WHEN <event>; one or more transition state} \\ \text{IF <condition>; conditions that must be met to trigger action} \\ \text{THEN <action>; one or more actions when the event takes place} \\ \text{VALID TIME <time period>; period of adaptation actions}\end{gathered} \tag{13.2}$$

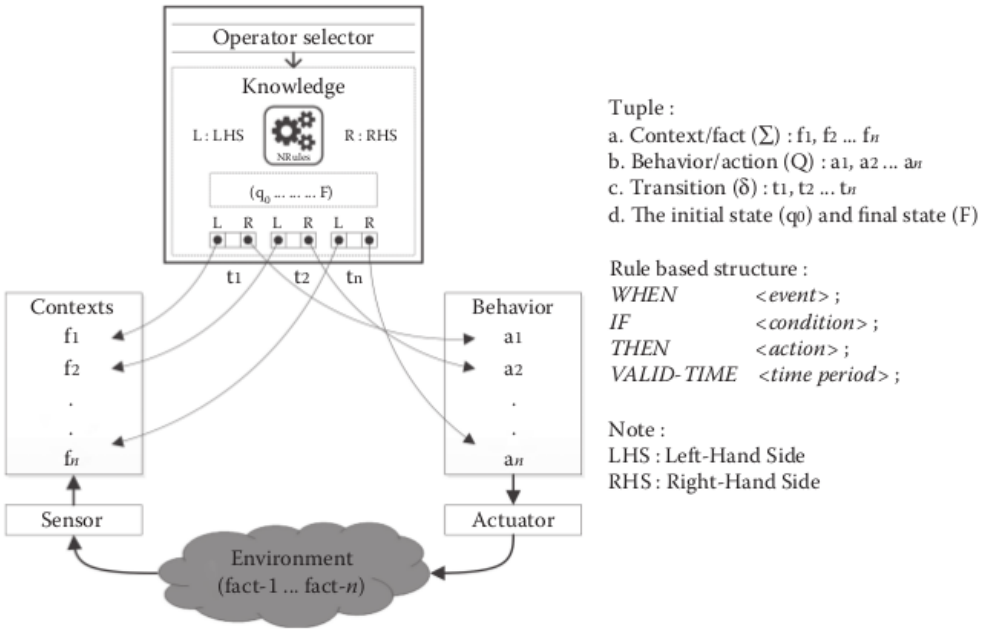

FIGURE 13.4
Model of rule representation.

Based on these descriptions, to define the needs of models, we set a tuple of the system, which are as follows: (a) Facts context information (∑: f1 … f*n*) as the set of inputs, (b) Action of system behavior (Q: a1 … a*n*) as a set of finite status, (c) Transition function or operator (δ: t1 … t*n*) as a function of the change, (d) Preliminary data (q0) as initial status, and (e) information targets status as the set of final status.

Figure 13.4 describes the relationship between variables and functions of the developed model. Information context is a set of inputs to be monitored as an event of the facts from any context information. The condition will be evaluated with reference to a specific event that occurs within the set of states, and it will trigger the adaptation action through the transition function of the prevailing condition; so the status of the target can be achieved by the action of the expected behavior.

Descriptions of the model are as follows:

1. ∑: Context information in the form of a set of facts (f1 … f*n*) is captured from the event environment and can determine the change in status.
2. Q: Behavioral actions (a1 … a*n*) form the set of behaviors status of the system, starting from the initial state (q0) to the status of which can be targeted (F).
3. δ: Transition function (t1 … t*n*) as time of action adaptation based on the evaluation of the condition, to change the initial status (q0) to the status of the target (F).

13.3.2 Structure of Knowledge Base

With reference to the representation of prepared rules, any change in the IoT system state can be expressed as series of a node changes (Supriana et al., 1989); each situation will be modeled into a node, and changes of state will be described as the arc which connecting

two nodes. So the used structure to handle the dependencies between the elements of knowledge is a directed graph.

The motivation for this development of the knowledge base structure is the reasoning that the approach of rule-based systems in most applications only focused on the rules of behavior of the software that is designed specifically, while the problems that may arise relate to the variability that can be served. This problem triggers the idea to formulate a model of the system structure to be more common and flexible, can be applicable to a wide scope of variability in the system and its environment, and expanded flexibly. The development of this model is targeted to meet the needs of the system to capture the variability in context and behavior of the system itself and its environment. This model can also set the criteria of assurance for the system management and its adaptation mechanisms.

In the design of the developed structures, each node in the graph has some slot categories, and each category has a number of attributes that can be increased, as shown in Figure 13.5.

The basic mechanism for knowledge management is prepared to meet the scope of a life cycle system: birth (create), use (use), updated (update), and deleted (delete). Developed-management module consists of initiation slot, expansion slots, adjustment slot, and inspection slot. The searching mechanism in a knowledge space is performed by using the access to the slot module, control of the trajectory, and the search delay. This mechanism can handle input consisting of context information, change and the output as behavioral adaptation actions of the sensors and actuator module. Overview model development can be seen in Figure 13.6.

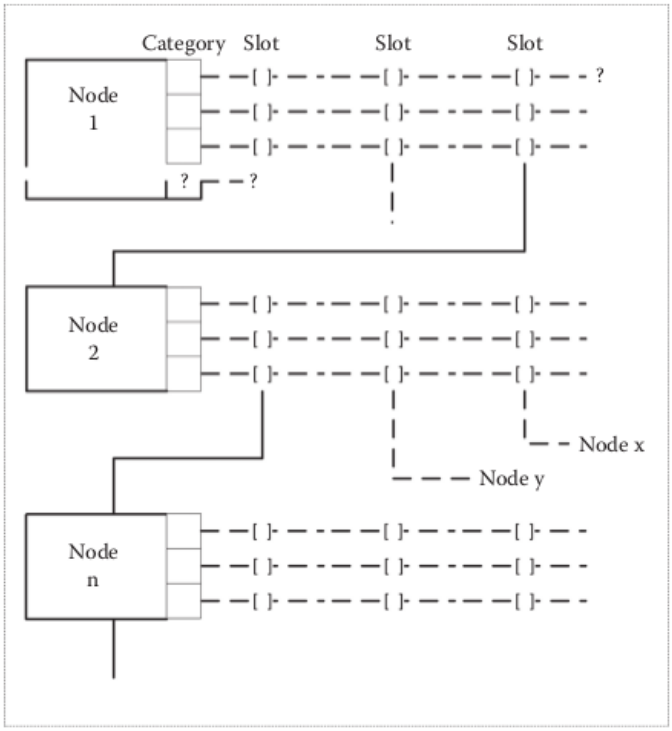

FIGURE 13.5
Structure of knowledge base.

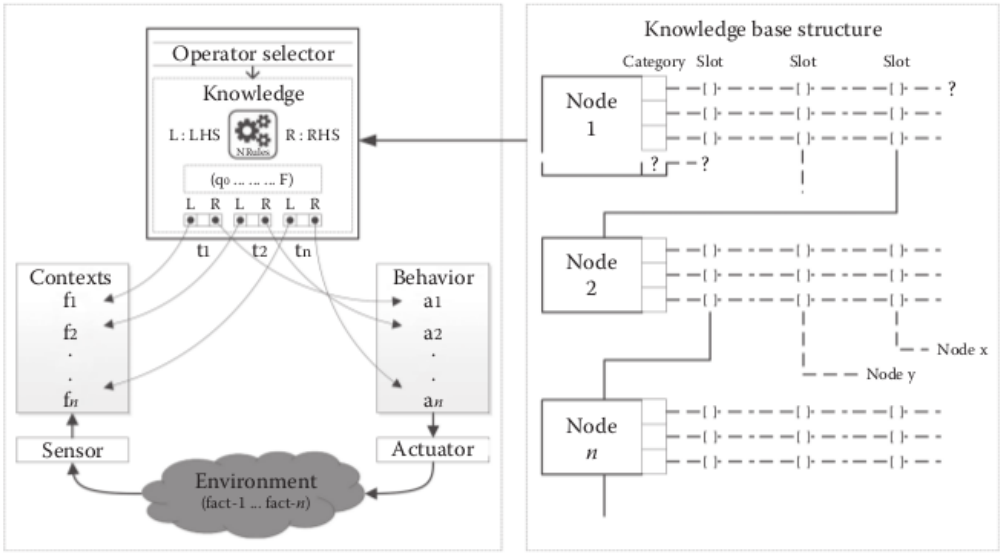

FIGURE 13.6
Representation and structure of knowledge base.

13.3.3 Reconfiguration Strategy

The main objective of this strategy is to define the domain model and determine the most relevant adaptation response options. The notation used to construct this algorithm consists of the following:

1. A model of goal (G) consists of connected nodes by its property attributes (P); each node also consists of a number of states (S*n*), which may have a contribution to a soft goal.
2. Some states (S*n*) may consist of the initial state (q0) and the status of the target (F); this state is influenced by several processing of facts (Σ: f*n*) on the LHS, which will determine the behavior of the action (Q: a*n*) and on the RHS, through the transition function (δ: t*n*)
3. The process of recognition performed by observing Σ f*n* as a trigger for S*n* in each G, until S can be set as a reference of plan preparation to realize a number of Q: a*n*, including its mapping to the component (C).

To realize the construction of the system and the need for changes, we define four basic operations (create, read, update, and delete). This operation is mapped to the operation of the components (Kramer & Magee, 2007) that can be seen in Chart 13.1.

In addition, we define the rule that represents each element in the components of goal systems. This rule is compiled based on the rules of goal decomposition (Nakagawa et al., 2008); if the goal decomposition is AND-decomposition, then the parent goal will require multiple attributes of the relation (port) for each child goal with a one-to-one degree of relationship. But if the goal decomposition is OR-decomposition, then the parent goal will provide the conditional attributes of the relation (port) to each goal, with the one-to-many degree of relation. In this activity, we also need to set the properties for each goal (functional) and a soft goal (nonfunctional). Chart 13.2 illustrates the algorithm to define a primitive component based on the goal model.

Chart 13.1 Configuration Algorithm

Components Configuration

```
G ← (goal, soft goal)
C ← (components)
for all N in goalModel do
    G ← C: configuration components for operation
    for each (∑, Q) ≠ ø do
        (create, read, update, delete) ← DiagnoseNode (G)
        create ← create component instance C from type G
        read ← connect or disconnect C1 to C2
        update ← set mode of C
        delete ← delete component instance C
        end for
    goalModel m ← reconfiguration(create, read, update, delete)
    enactModel(m)
end for
```

Chart 13.2 Diagnosis Algorithm

Goal Diagnosis

```
for all G in m do
    G ← addProperty(P)
    if G decomposition = AND-decomposition then
      G parent ← add multiplePort // sum of G childrenPort
      for all G in G children do
          port ← add providedPort
          C configuration ← DiagnoseNode
        else if G.decomposition = OR-decomposition then
            G parent ← add conditionalPort
            for all G in G children do
              port ← add conditional providedPort
              C configuration ← DiagnoseNode
            end for
        end if
      end for
    end if
end for
```

The development-control strategy to observe and regulate every component of the system that has been defined, by using the design pattern (Abuseta and Swesi, 2015) that was inspired by the model of MAPE-K (IBM, 2005), and modified in accordance with the previous system requirements. Chart 13.3 shows the algorithm to monitor and analyze the needs of the adaptation plan.

Chart 13.3 Observation Algorithm

States Observation

```
for all G in m do
    m ← (∑ fn) // at run-time
    G ← getValue(P) // time triggered or event triggered
    for each value(S) in P do
        S ← combining internal and external value(P)
        if S in S.target(F) ≠ P.threshold then
            systemState ← new S.system(S.instance) and
            systemStateLog ← save(S.system) and
            send information(S.system) to analyzerManager
        end if
        for each S.system in analyzerManager do
            analyzer ← update(logs) actual S.system
            search(S.system) in symptomRepository
            if symptom ≠ ø then
                create(adaptationRequest) and
                update(adaptationRequest) for plan specification
                else
                    addSymptom to symptomRepository and
                    create(adaptationRequest) and
                    send information(adaptationRequest) for plan specification
            end if
        end for
    end for
end for
```

There are a number of properties on the model of goals that must be read and measured (observation) in concurrency. In the Java programming language, multithreading techniques are relevant to meet this need. This activity represents the state system at run-time by using time-triggered or event-triggered; this process is performed in response to any request or event. The state of the system at run-time is represented by a combination of the internal and external property value; the desired state is directed by a goal and a soft goal. Violations of the state detected by the threshold level of each goal property and a new run-time system state are stored in the system state logs for analysis.

Violation of the goals and requirements of the system is analyzed based on the symptom repository. The symptom repository is a collection of symptoms that has been set for the system to avoid and heal itself. This component is part of the knowledge base. The symptom repository is equipped with facilities to add new symptoms that occur during runtime analysis through operating add symptom. The symptom class consisting of an associative array is used to store any symptoms, which represent each event of symptoms and its value represents the condition relating to the event. Some programming techniques for this class and interfaces are (Abuseta and Swesi, 2015) map (Java), dictionary (Python), and associative arrays (PHP). If the analysis detects the presence of symptoms, then the plan component will receive a demand signal for adaptation and reconfigure the system based on the engine policy. Chart 13.4 shows the algorithm reconfiguration plan.

The policy engine provides the high-level goals that control the operation and functions of related systems. The general form is ECA rules; these rules are used to determine the action when the event occurred and meet certain conditions. The policy engine is represented as a knowledge base, which provides an interface for system administrators to determine and change the system policy. In our version, the engine is being expanded with the model of rule editor, and also additions or changes in the specification can be performed by editing the knowledge base directly, or reset back into the system.

Chart 13.4 Reconfiguration Algorithm

Reconfiguration Plan

```
for each adaptationRequest(S.system) do
    init ← set work (∑, Q)
    while δ {t_n (f_n, a_n) | n ≠ ø} do
        δ ← find that the LHS of the operator match with work say it found
          if found is only one then
              RHS ← set work
              else if work is equivalent with target then
              stop succeed
        end if
        if found is more than one then
              found ← set work one of the found
              backs ← put rest of found
        end if
        if found is empty then
              else if backs is empty then
              stop failed
              else
                  backs ← set work one of backs
              end if
        end if
    for all δ(t_n) is found do
        a ← construct correctiveAction(addAction)
        changePlan ← newChangePlan(a_n)
        send changePlan to one or more executors
        for each a in executor do
              actuator ← update(a_n) // one or more actuators
              S.system ← reconfiguration m with actuator
              // set new value for C (DiagnoseNode)
              systemStateLog ← saveState(S.system)
        end for
    end for
end for
```

Each request of adaptation will be represented as a state in the system, which detected based on the occurred events. A set of δ can be expressed as δ {t*n* (f*n*, a*n*) | $n \neq \emptyset$}, with "f*n*" is the fact of context (the property of goal) and "a*n*" is the behavior of the action that

expected to a specific contextual n. The quality of the inference engine depends on the selection of state for the adaptive action on the RHS class. This quality is also determined by the expression in the LHS class. The expression is a match between the rule and the facts. For example, comparing two or more collections of properties with certain structures, based on the required criteria. The used key strategies are the forward strategy, reusing (reuse) the existing fundamental component, and matching the required specifications.

The changing plan contains the set of action for the execution component to perform the adaptation action. This action should be carried out in some specific order, for example, sequentially or concurrently or both. The execution component will use a number of actuators to set new values of the property in the target system and its environment. The corrective action of the execution component on a number of the actuator will bring the system back to the desired state or an acceptable state, and then this state will be saved in the log of the system state by the actuator.

13.3.4 Component-Based Software

In this section, a description of the reconfiguration strategy that has been discussed in Section 13.3.3 will be implemented in the component-based software. The component is a specific function unit which interacts with using the interface through the delivery of a service. The component-based system is a collection of components which are connected to each other, so it has the desired functions. In general, the interface between the components is a standard form, so the components can be connected and have a good interoperability. Based on the variety of this interface, the components can be divided into several types, namely (a) the input component, (b) the process components, (c) output component, and (d) the delivery components.

The input component is a component that only has an interface that generates information. The process component is defined as a component which has a pair or more connections between the input interface and an output interface. In general, information for the output interface in the components of the process have a functional relationship with the information on the input interface. Some of the functions can be owned by any other process components, including structural, preparation, grouping, and counting function.

Output component is a component that only has the interface that receives the output information from the processing component. And the delivery component is a supporting component that mediates the delivery between the input component and output component. By using this delivery component, the system can become more compatible to be accessed in the variety of platforms; it is mostly needed in a system with the concept of IoT.

Each component can provide independent services, connected or collaborated services, and free active services. This independent service allows a component to perform its function itself, actively, without requiring the presence of other components. The components with the connected services or collaborated services are the component with the interaction of two or more components connected. In particular, this process can take place sequentially or in parallel, and the component of the free active services is a component that is active and running but has not performed its function until there are triggers that appear from any other active component.

One of the approaches that can be adopted to formulate the architecture of the software components, for the needs of a cyber-city system in the concept of IoT, is to utilize the architectural description languages, which was developed in the Darwin model (Magee et al., 1996; Magee et al., 1996; Hirsch et al., 2006). This model is a declarative

component-based architectural description languages that supports a hierarchical model and graphical modeling, easy to use in determining the software components, including interconnection and structure by using formal modeling notation. The specification of this software component architecture will be incorporated into the requirements of the self-adaptive cyber-city system in Section 13.4, as a motivation of the case to provide a more concrete.

13.4 Case Study of Cyber-City System

This section discusses the case about the development of the cyber-city system in the IoT concept; the goal of this case is to make the cyber-city system with the ability to capture requirements of each element of a city and has a good adaptability to changes and have the capability to handle the growth of the system in its environment.

13.4.1 Modeling of System

The development of the cyber-city system can be initiated through the domain modeling in the IoT concept; the system will identify each entity until the processes or resources can be defined, and the relation between each other can be explained. Each of these processes is connected to each interface that serves as executor of the system. This activity can be done with the goal-based modeling approach. Furthermore, each defined element in the system is mapped into the software components, until the operational mechanism of the system can be determined, whether the components are set to dynamically or statically.

The stages are performed to identify and define the IoT context of adaptation mechanisms; so in every identified context, we can build a knowledge tree, as shown in Figure 13.7. As an example, according to the context of the user interface identification mechanism, the structure of knowledge can anticipate the growth of user requirements, both of internal or external stakeholders. In another example, with the basic mechanism of service-context identification, the knowledge structure can anticipate the changes and growth of requirements on the activity and the process of converting data into system services. In addition, it is possible to add an identification mechanism of other contexts if necessary, based on the changes of facts or the growth of the system.

The illustration in Figure 13.7 shows that the structure of actor tree and data tree will be automatically added, deleted, updated, and adapted to every vertex, in response to changing circumstances and growth occurs. As an illustration, every actor tree in the cyber-city system will be linked to the requirements and changes of actors in government, private, personal, and so on, as a user of the system. Data tree will be associated with the process of change and growth activity of each actor in the service system, whether it is in the process of trade, education, transportation, health, tourism, entertainment, and so on. Based on the formulation of this knowledge-tree structure, we can define the requirements of PSn in the form of the service catalog as shown in Table 13.2.

The service catalog describes details of the service for each actor. Duration is the service availability of time, for example, 2 × 24 hours, 8:00 a.m. to 4:00 p.m., and so on. Options are a class of selected service based on the level of interest or influence to the goal, for example, level 1: A response (1-2), level 2: B response (2-3), and level 3: C response (3-4). Moreover, the

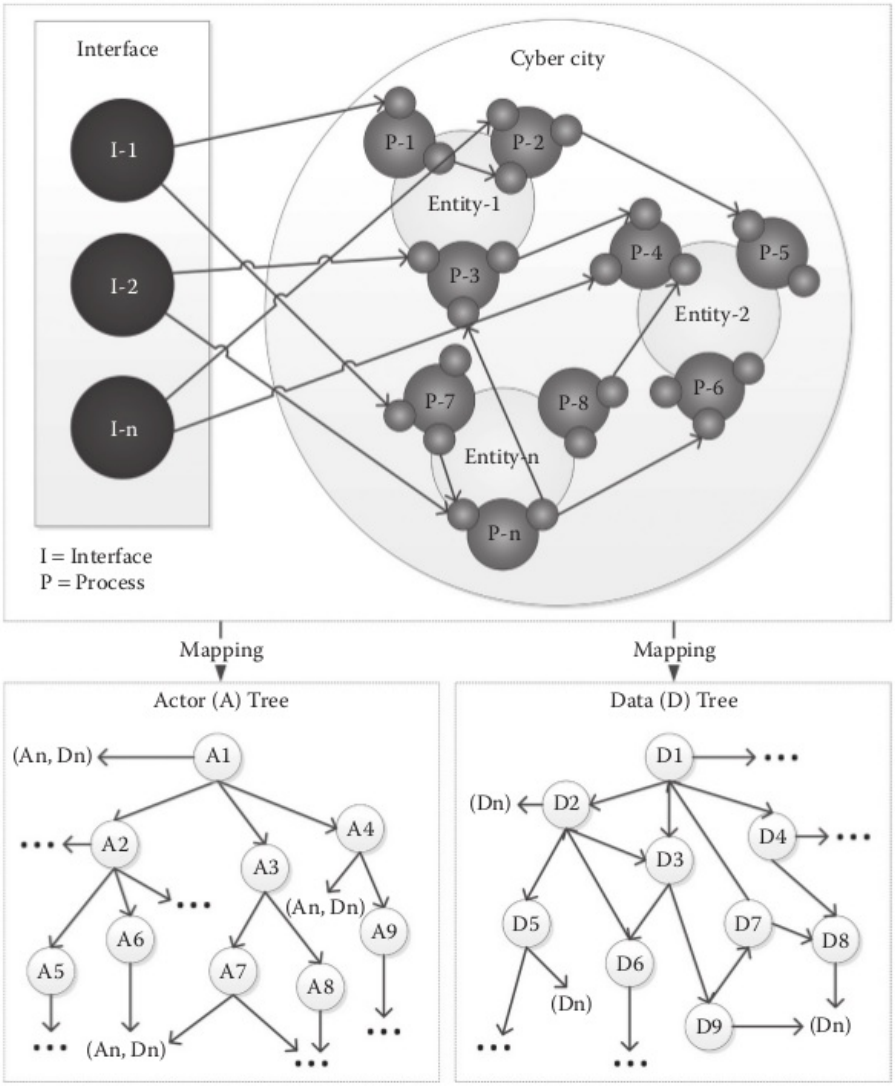

FIGURE 13.7
The mapping of elements in a cyber-city system into the knowledge tree.

TABLE 13.2
Service Catalog

	Description			User			
Service Catalog	**Duration**	**Option**	**Response**	U_1	U_2	U_3	U_n
Service-1.1	24 × 7	2	2	√	–	√	√
Service-1.2	24 × 7	2	1	√	–	√	–
Service-1.n	24 × 7	1	1	–	√	–	√
Service-2.1	24 × 7	1	2	√	√	√	–
Service-2.2	08–16	3	4	–	√	√	√
Service-2.n	08–16	1	1	√	√	–	–
Service-n.m	24 × 7	3	3	–	–	√	√

response is the incident recovery activity of the service class, which is classified based on the level of seriousness of the incident, for example, level 1: 10 min, level 2: 30 min, level 3: 1 hour, and level 4: 2 hours, depending on the impacts of incidents.

The changes that can lead to incidents in the service catalog can be categorized as the facts of (f*n*) changes, for example:

1. f1: The political policy of the city can be associated with the changes in infrastructure and the structure of the city departments.
2. f2: The growth of activity, regarding changes to business processes and growth of PSn.
3. f3: Service providers, changes legalization, are caused by the demand for a new form, features, and algorithms, in the application of external services.

The category of service changes can be considered to set the service catalog for strategies, management, and recovery. Request for a fast response to make changes in the system can be made by performing a matching process between the occurred facts against the facts in the structure of the knowledge tree. The service catalog consists of business-process service which contains the entire PSn and technical service (Tsn). Tsn relates to all technical specifications of the technology. Figure 13.8 shows the adaptation mechanisms from capturing events by the agent from the fact (f1, f2, …, f*n*) in the system environment. The agents also capture the context of information from the service catalog (PSn, Tsn) until the action implementation.

An event is the information of any context which is connected with the specification of the new facts inside environmental and service catalog ($\sum$ = f1, f2 … f*n*; PSn, TSn). The created condition will evaluate the situation in accordance with the occurred specific events, including the captured characteristics of the service and the changes in every context (Q = f*n* ←→ PSn, f*n* ←→ TSn). So the evaluation of the service level agreement (SLA) is performed to select the most appropriate action behavior.

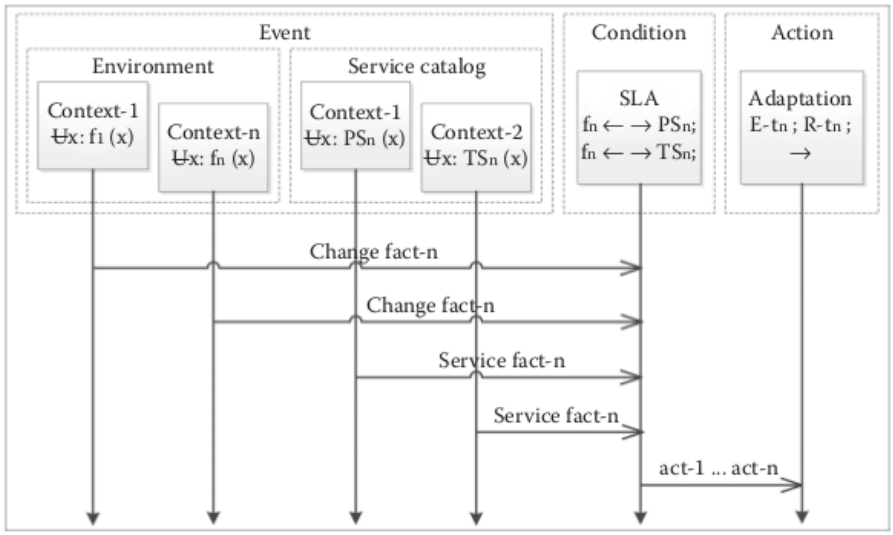

FIGURE 13.8
The dynamic behavior of the system.

Based on the prevailing condition, the action of adaptation is performed (δ = *E*–t*n*; *R*–t*n*). The reconfiguration process of evolution (*E*–t1, E–t2, … *E*–t*n*) at the time (t) made upon the consideration of the SLA attainment targets evaluation, although the action of reconfiguration (*R*–t1, *R*–t2, … *R*–t*n*) or the handling changes that have not been stipulated in the SLA. Basically, this PSn adaptation action deals with the authorization of the service portfolio, for example, when the service should be updated, replaced, maintained, refactored, dismissed, and rationalized. All of these services represented adaptation actions within the control of changes to the structure of the tree of knowledge.

13.4.2 System Configuration

The mechanisms are developed to manage the service catalog, as shown in Figure 13.9. The modeling process uses the TAOM4E tools (TAOM4E n. d.). A goal of the cyber-city system services is managed using three subgoal activities such as the user interface, service application, and service providers. Each goal can have subgoals and also plans that contribute to the soft goal.

The modeling services through the goals element, as shown in Figure 13.9, are mapped into the software components as shown in Figure 13.10. Each goal and plan will form a primitive component so that there are four component groups—3 groups serve to detect the context and 1 group as the reconfiguration components. The three groups are "detect context-1" for the user interface, "detect context-2" for the service application, and "detect context-3" for the service provider. All three groups of these components require a group of components that has the function to perform the reconfiguration; the

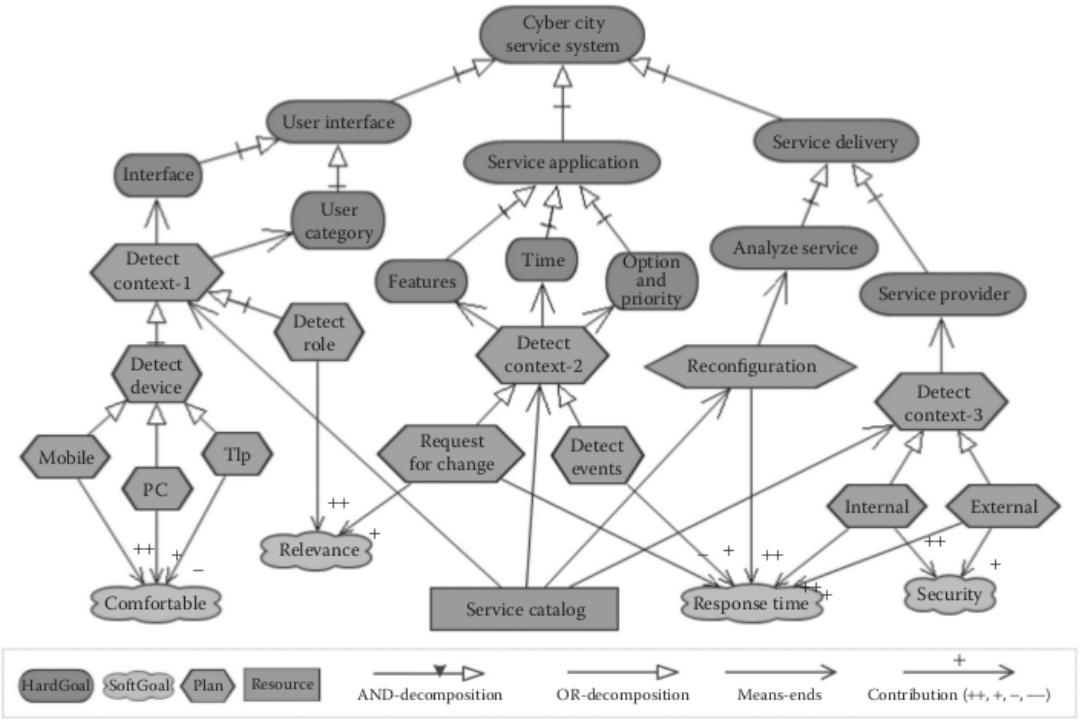

FIGURE 13.9
The service catalog management for a cyber-city system.

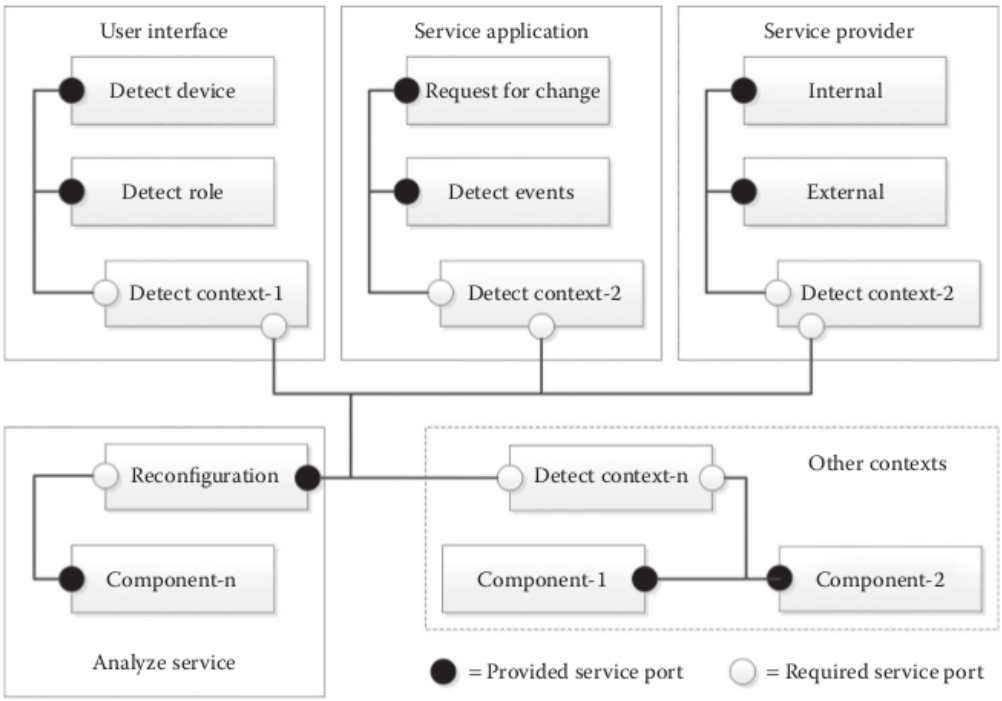

FIGURE 13.10
Software components configuration.

reconfiguration components will determine the adaptation strategies for each component that needs it, so the presence of this component is generic. That means if at run-time systems require the addition of new components, then automatically, these components can be added as shown in the dashed line in Figure 13.10. It is an example of adding a new component to detect new context. Likewise, if a component is not needed or need to be changed or need to be reused, it can be executed at run-time.

The request for change component is primitive for the component that requires reconfiguration. The model is initialized by the variable the component dynamic, which is determined by conditions at run-time. Achievement of the goal-service application consists of features, time, options, and priority. This achievement is the target of the component. This target is activated based on the soft-goal criteria and constraints. The soft goal is a consideration of the contribution, against the nonfunctional system, whereas constraints are the domain assumptions, which are set based on the strategy reconfiguration as described in Section 13.3. The components of "Detect Context-2" is a composite component that describes the primitive component interconnect and needs with the component reconfiguration through the type of OR relations in accordance with the modeling goal.

Based on the configuration of developed components, each component will establish a knowledge according to the needs of their respective functions. Figure 13.11 will illustrate the knowledge tree from the results of software components operations. For example, the user interface and its goal plan "detect context-1" will form a tree of actors or users, whereas the goal service application with the plan "detect context-2" will form a tree of data or services, so does the other goals and plans.

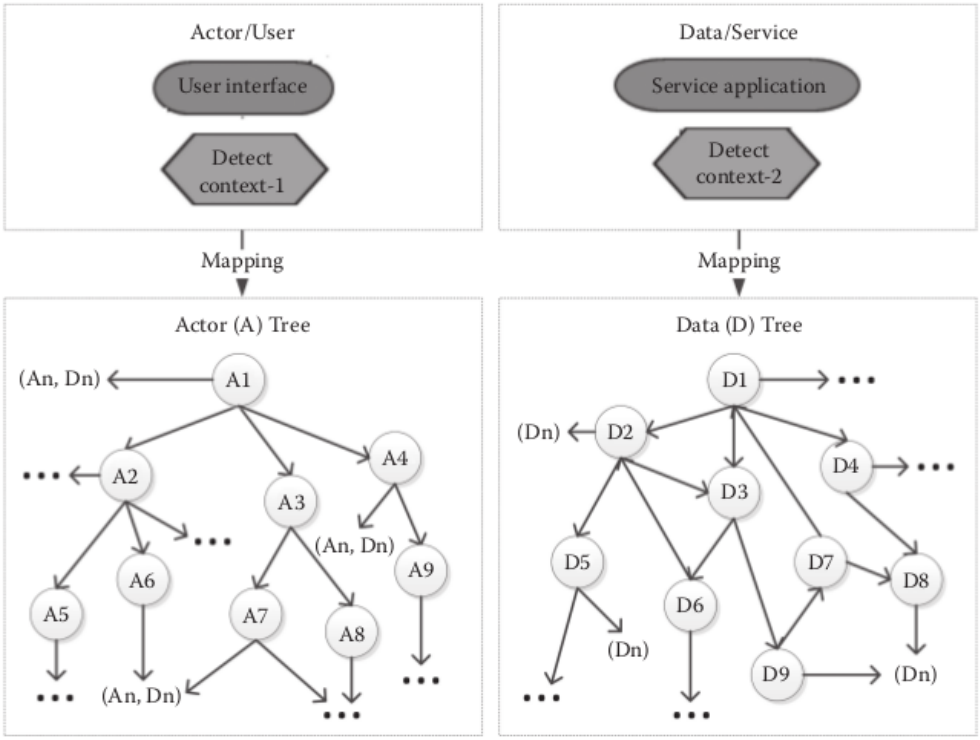

FIGURE 13.11
The mapping of components into the knowledge tree.

13.5 Discussion and Conclusion

13.5.1 Discussion

SAS is a research area that has a broad spectrum because it deals with a variety of factors and various conditions that must be recognized. Discussion performed by limiting the scope of the basic goal-oriented in requirements engineering approach.

In this article, we introduced an approach that is based on the goal-oriented in requirements engineering approach by inserting the additional elements through the primitive construction of the system as discussed in Sections 13.3 and 13.4. Figure 13.12 shows the overall architecture models, which can be used as a guide for developers in constructing a SAS in general, and self-adaptive cyber-city system in the IoT concept in particular.

Description of the system-development model in Figure 13.12 is as follows:

1. The development activities begin with the goal modeling to represent the IoT domain model. The available tools at this stage can be adjusted with an adopted goal-model approach, for example:
   a. TAOM4E (TAOM4E n. d.) to model Tropos (Bresciani et al., 2004)
   b. Objectiver (http://www.objectiver.com/) to model KAOS (Dardenne et al., 1993)

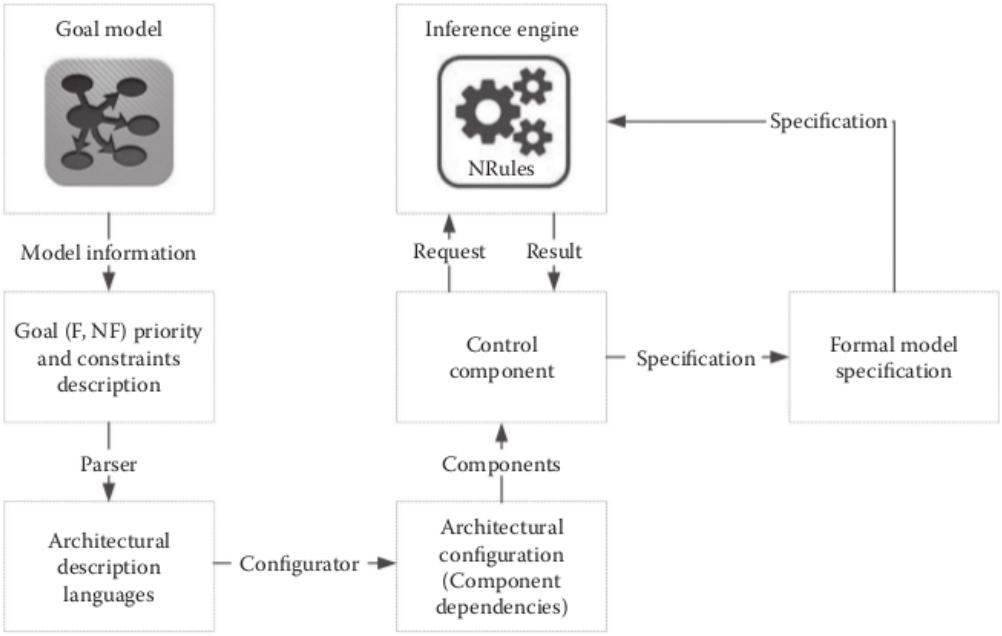

FIGURE 13.12
Development model of a system.

c. Open OME (http://www.cs.toronto.edu/km/openome/)
d. STS-ml (http://istar.rwth-aachen.de/tiki-index.php?page=i*+Tools)
e. OmniGraffle (http://www.omnigroup.com/omnigraffle), and others

   Or we can arrange a description of this goal by developing its metamodels, for example, using EMF (http://www.eclipse.org/modeling/emf/updates/ or http://www.eclipse.org/modeling/emf/downloads/).

2. By the time of preparing the description of these goals, we need to ensure that any functional (F) requirement should be represented as a goal and nonfunctional (NF) requirements as soft-goals that can reflect the quality criteria of the system. How to determine the parameters of the priorities and constraints are discussed in Section 13.3.
3. After mapping the description of the goals that have been made in the architecture description language, we configure it into a software component. This is accomplished by using a software to generate the required components.
4. The preparation of the software component architecture can refer to Section 13.3.4. The supporting code generator tools such as java emitter templates (JET) Template (http://www.eclipse.org/modeling/m2t/downloads/) can be used to transform the eclipse modeling framework (EMF) meta-model. Or some other tools mentioned in point a, which has been equipped with the code generation facility.
5. The component control will coordinate all the components to give the inference function. This component generates formal model specifications to perform the selected reasoning tasks.

6. The components of inference engine can be developed by drafting rules, as discussed in Section 13.3.3. This rule-based system can also be modeled using the EMF meta-model tools. Some of the tools that can be used as an engine rule are CLIPS (http://www.ghg.net/clips/CLIPS.html), JESS (http://www.jessrules.com/jess/download.shtml or http://herzberg.ca.sandia.gov/jess/), jDREW (http://www.jdrew.org/jDREWebsite/jDREW.html), Mandarax (http://mandarax.sourceforge.net/), or using the Datalog Rules and DLV inference engine (Leone et al., 2006), and others; they can be adapted to the description of the requirements of the goal.

The selection of tools that will be used in this IoT system development must be definitely determined by studying the alignment between the description of goals and the needs of the inference engine by considering any capabilities of tools to cover all requirements from the developed concept.

13.5.2 Conclusion

Characteristics of the cyber-city system have much diversity of elements and are very dynamic; it is caused by the very quick growth of the environment. The proposed solution in this article is formulated through two approaches. First, importance to understand and capture the variability in context (IoT concept) and behavior of the system. This is realized through the domain modeling context with the logic-based approach. Second, the availability of the knowledge structure and the quality of the inference engine that can handle the scope of more broad and flexible solution space. This is realized through the modeling of the inference context with the rule-based approach.

The main highlight of discussion in this article is the provision of the SAS to meet the scope of a system life cycle in the IoT concept; this relates to the control of automation which is the part of the main concept of IoT. The given idea is an opportunity that can be followed up in the control of automation of the future of the IoT. So the concept of IoT system is expected to become more aware and adaptive; all orders can be performed automatically, organized, intelligent, acting independently in context and situation, or event environment.

The adaptation strategies as a mechanism for reconfiguration are proposed to overcome this problem; a series of modifications were developed to represent the problem space. In addition, this strategy also prepares the system to be able to conduct a search of the solution space as an appropriate alternative solution automatically in the implementation of the control strategy. The developed concept was inspired by the seven pillars of taklif systems which are the representations of life in the universe. These pillars are formulated into the rules of systems with self-adaptive capabilities. We believe this proposed concept will contribute to the provision of software systems in the context of a dynamic environment, and especially in the construction of the cyber-city systems in the IoT concept. We expect this concept to accommodate any issues of diversity and dynamism inside the system and its elements.

References

Abuseta, Y. and K. Swesi. Design patterns for self adaptive systems engineering, *International Journal of Software Engineering & Applications* (*IJSEA*), 6(4), pp. 11–28, 2015.

Agustin, R. D. and I. Supriana. Model Komputasi Pada Manusia dengan Pendekatan Agent, Kolaborasi BDI Model dan Tujuh Pilar Kehidupan sebagai Inspirasi untuk Mengembangkan Enacted Serious Game, *Konferensi Nasional Sistem Informasi* (*KNSI*), ITB, 2012.

Aradea, I. Supriana, and K. Surendro. An overview of multi agent system approach in knowledge management model, *International Conference on Information Technology Systems and Innovation (ICITSI)*, IEEE, School of Electrical Engineering and Informatics, ITB, 2014.

Bratman, E. M. *Intentions, Plans, and Practical Reason*, CSLI Publications, Stanford, CA, 1987.

Bresciani, P., A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos. TROPOS: An agent-oriented software development methodology, *Journal of Autonomous Agents and Multi-Agent Systems*, 8(3): 203–236, 2004.

Cheng, B. H. C., K. I. Eder, M. Gogolla, L. Grunske, M. Litoiu, H. A. Muller, et al. Using models at runtime to address assurance for self-adaptive, In *Model@run.time Foundations, Applications, and Roadmaps, Lecture Notes in Computer Science* (*LNCS*), Vol. 8378, pp. 101–136, Springer, 2014.

Daniele, L. M. Towards a rule-based approach for context-aware applications, Master Thesis, University of Cagliari, Italy, 2006.

Dardenne, A., A. van Lamsweerde, and S. Fickas. Goal directed requirements acquisition, In *Selected Papers of the Sixth International Workshop on Software Specification and Design*, pp. 3–50, Elsevier Science Publishers B.V., Amsterdam, 1993.

Ganek, A. G. and T. A. Corbi. The dawning of the autonomic computing era, *IBM Systems Journal*, 42(1): 5–18, 2003.

Hirsch, D., J. Kramer, J. Magee, and S. Uchitel. Modes for software architectures. In V. Gruhn and F. Oquendo (Eds.), *EWSA 2006, Lecture Notes in Computer Science* (*LNCS*), Vol. 4344, pp. 113–126, Springer-Verlag, Berlin, Germany, 2006.

IBM, *An Architectural Blueprint for Autonomic Computing*, IBM, Hawthorne, NY, 2005.

ITU-T, Overview and role of ICT in smart sustainable cities Telecommunication Standardization Sector, 2014.

Kramer, J. and J. Magee. Self-managed systems: An architectural challenge, *Future of Software Engineering, FOSE'07*, pp. 259–268, The ACM Special Interest Group on Software Engineering (SIGSOFT), IEEE Computer Society, Washington, DC, May 2007.

Leone, N., G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic* (*TOCL*), 7(3): 499–562, 2006.

Magee, J., N. Dulay, S. Eisenbach, and J. Kramer. Specifying distributed software architectures. In *Fifth European Software Engineering Conference* (ESEC95), Barcelona, September 1995.

Magee, J., J. Kramer, and D. Giannakopoulou. Analysing the behaviour of distributed software architectures: A case study. In *5th IEEE Workshop on Future Trends of Distributed Computing Systems*, pp. 240–245, 1996.

Mora, M. C. BDI models and systems: Reducing the Gap. In J. P. Muller et al. (Eds.), ATAL'98, LNAI 1555, pp. 11–27, 1999. Springer-Verlag, Berlin, Germany, 1999.

Nabulsi, Dr, R. M. *7 Pilar kehidupan: Alam semesta, akal, fitrah, syariat, syahwat, kebebasan memilih, dan waktu*, Gema Insani, Jakarta, 2010.

Nakagawa, H., A. Ohsuga, and S. Honiden. Constructing self-adaptive systems using a KAOS model, In *Proceedings of the SASOW*, pp. 132–137, IEEE, 2008.

Pires, L. F., N. Maatjes, M. van Sinderen, and P. D. Costa. Model-driven approach to the implementation of context-aware applications using rule engines, In M. Mühlhäuser, A. Ferscha, and E. Aitenbichler (Eds.), *Constructing Ambient Intelligence. AmI Workshops. Communications in Computer and Information Science*, vol. 11, pp. 104–112, Springer, Berlin, Germany, 2008.

Russell, S. and P. Norvig. *Artificial intelligence, a Modern Approach*, 2nd Prentice Hall, Upper Saddle River, NJ, 2003.

Supriana, I. and D. Aradea. Model self-adaptive sebagai landasan sistem untuk menunjang penumbuhan komunitas, *Keynote Paper Seminar Nasional Teknologi Informasi dan Komunikasi* (*SENTIKA*), Vol. 6, Yogyakarta, Maret 18–19, 2016.

Supriana, I. and D. Aradea. Automatically relation modeling on spatial relationship as self-adaptation ability, *Internationl Conference on Advanced Informatics: Concept, Theory and Application (ICAICTA)*, Vol. 2, IEEE, Bangkok, Thailand, 2015.

Supriana, I., S. Wahyudin, and A. Mulyanto. Pengembangan motor inferensi untuk aplikasi sistem pakar dalam model diagnosa, Technical Report, School of Electrical Engineering and Informatics, Bandung Institute of Technology, Kota Bandung, Indonesia, 1989.

Tool for Agent-Oriented visual Modelling for Eclipse (TAOM4E) and its plugin t2x (Tropos4AS to Jadex), developed by the Software Engineering group at Fondazione Bruno Kessler (FBK), Trento, available, including the extensions, at http://selab.fbk.eu/taom.

Wu, C. G. Modeling rule-based systems with EMF, Eclipse Corner Article, Copyright (c) 2004 Chaur G. Wu. All rights reserved, 2004.

Paper 4

ORIGINALITY REPORT

8%	2%	1%	5%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

MATCH ALL SOURCES (ONLY SELECTED SOURCE PRINTED)

5%

★ Submitted to Universiti Teknologi Malaysia

Student Paper

Exclude quotes	Off	Exclude matches	< 1%
Exclude bibliography	On		