VOL. 10, NO. 3, FEBRUARY 2015 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences

www.arpnjournals.com

FORMAL SPECIFICATION OF SOFTWARE DEVELOPMENT USING BUSINESS RULES APPROACH

Yaya Setiyadi, S. Si, M. T.[1], Oerip S. Santoso[2], Cecilia E. Nugraheni and ST. MT[3].
[1]Del Institute of Technology, Toba Samosir, SUMUT, Indonesia
[2]Teknik Informatika - ITB, Bandung, Indonesia
[3]Parahyangan University, Bandung, Indonesia
E-Mail: yaya@del.ac.id

ABSTRACT

An organization define rules in its business to achieve the organizational goals. Along with developments and changes facing the business environment, often an organization's business rules have to change. Those changing can infer the software system of the organization. This could lead to an inefficient and ineffective system changing, if the organization has to change the whole system, for every small change of the business rules. This paper discus one approach to separate the business rules from other components of the system, as the changing of the business rules will not influence others components. The approach is called the Business Rules Approach. The approach will be implemented in a case study, the registration process of the academic information system of Del Institute of Technology. In order to check the correctness of the specification, the specification of the system with business rules approach is represented formally, implementing RAISE methodology and using notation of the RAISE Specification Language (RSL). The formal specification has been successfully verified and meets the requirement of the implementation relation criteria.

Keywords: Formal specification, verification, business rules approach.

INTRODUCTION

An organization define rules in its business to achieve the organizational goals. These rules are usually applied in the application of organizational software systems that is built to support the passage of an organization's business processes. Viewed from the perspective of software development, identifying the rules are part of requirements analysis process, which in turn will affect the design and implementation of applications. The rules and the organization's business processes are embedded into the application software as illustrated in the Figure-1.

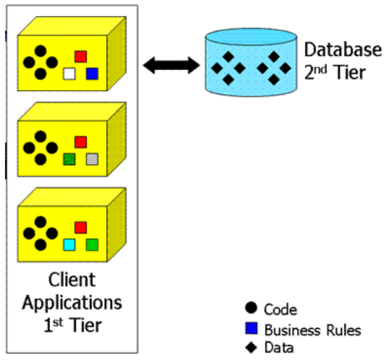


Figure-1. Hard coded business rules.

Along with developments and changes facing the business environment, often an organization's business rules have to change. Here then problems arise, because of the merging of the organization's business rules and processes into the application system, changes in business rules will cause major changes in the system that has been built, especially if new rules are not in accordance with the design of software that has been built, so the system must be reengineered. Surely this will require no small effort and a long time. The more frequent changes occur within a short period of time, then the issue will be increasingly difficult to resolve.

To resolve the above issues, a system that can adapt to changing business rules organization is needed (changeable system). One solution is offered by separating business rules from other aspects of the system. By separating the business rules of the system, and handle it as a separate asset, optimization of the rules and rule changes can be done without making changes to other aspects of the system.

Separation of business rules from other aspects have been submitted by Ronald G. Ross (Ross, 2003) in his book “Principles of the Business Rule Approach” and Barbara Von Halle (Halle, 2002) in her book “Business Rules Applied, Building Better Systems Using the Business Rules Approach”. Ross and von Halle offers an approach in building a system with more emphasis on system development orientation of business rules (rules orientation) in order to facilitate its treatment in the future. The system architecture with the business rules approach can be seen as in Figure-2.

Furthermore, to ensure the correctness of the separation of business rules on the approach, we need a verification process, both of these rules also to the separation. So far there has been no literature which specifically discusses how to perform formal verification against business rules separate from other aspects of software development. This paper will discuss how to perform the separation of business rules and how to conduct its formal verification.

VOL. 10, NO. 3, FEBRUARY 2015 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences

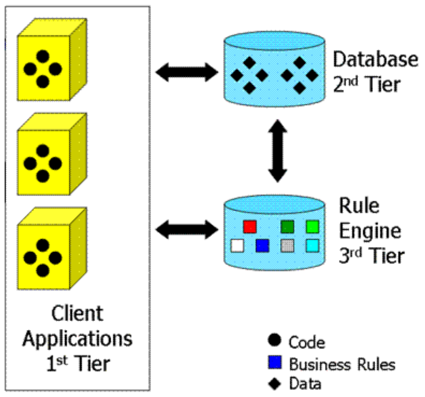


Figure-2. Business rules approach.

SUPPORTING THEORIES

Definition of business rules

Ronald G. Ross (Ross, 2003) stated that, at the theoretical level, rule based directly on the predicate calculus. More specifically, the rule is IF-THEN connective on the predicate calculus, known also as the logical connective implication or implication. The general form of IF-THEN connective is:

$$IF \;\; p \;\; THEN \;\; q \qquad (1)$$

Where p and q must be a Boolean expression, namely that they must have truth values. p is called the antecedent, and q is called consequent. The form above can also be read as p implications q.

Business rules classification

Classification of business rules vary depending on the purpose of presentation for different users. Von Halle summarizes some of the classification into a scheme, the business rules consist of three components: Terms, Facts, and Rules, with a brief description as follows:

A term is a noun or noun phrase that was agreed to have a specific definition. It will be referenced by other types of business rules. More briefly, the meaning of terms is a value, variable, and function.

A fact is a statement that connect or relate some of the terms, using a preposition and a verb phrase. The fact also be referenced by other types of business rules.

A rule is a declarative statement that applying the rules of logic or computation on the values of information. Results of a rule may be the discovery of new information or a decision to take an action. Von Halle (Halle, 2002) classify rule into: constraint , guideline, action-enabler, computation and inference, as shown in Figure-3.

The business rules approach

Von Halle (Halle, 2002) states that the business rules approach is a methodology in which the business rules expressed and presented (capture), tested, publicized, otomize, and amended in light of its business strategy. The results of this method is a business rules system, a system where the rules are separated, the logical and perhaps even physically, from other aspects of the system and dividing by the data storage, user interface, and applications.

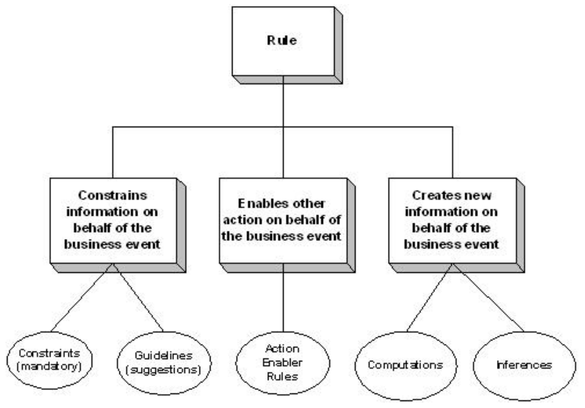
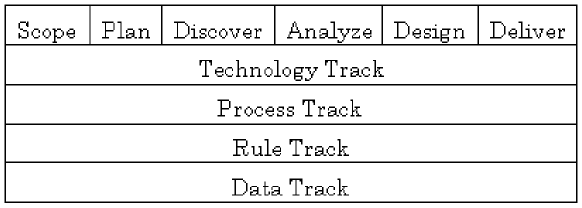

Rule
Constrains information on behalf of the business event
Enables other action on behalf of the business event
Creates new information on behalf of the business event
Constraints (mandatory)
Guidelines (suggestions)
Action Enabler Rules
Computations
Inferences

Figure-3. Rule classification scheme.

The business rules approach based on the following four principles, which was shortened by STEP:

(S) Separate. This means that we do the separation of the rules from other aspects of the system.
(T) Trace. This means that we handle every rule linkage of the two directions. The first direction is through the origin / source of the rule. The second direction is tracking the implementation of these rules.
(E) Externalize. This means that the rule is expressed in a form that can be understood by those non-technical business party.
(P) Position. This means that the rules are always positioned in a condition such that easy to make changes to it.

A business rule approach has six distinct phases, namely scope, plan, discover, analyze, design, and deliver, as shown in Figure-4.



Scope	Plan	Discover	Analyze	Design	Deliver
Technology Track					
Process Track					
Rule Track					
Data Track					

Figure-4. Business rules approach phases.

RAISE methodology

RAISE is a software engineering methods, so the stages of development have in common with software engineering development methods in general. Stages of development in RAISE is a process of analysis, design, verification and translation.

As a starting point for the analysis phase, the method can be used object-based software development. Stages of development on the basis of the object begins with an analysis of the use case diagram, then proceed with making the use case scenario, state-transition diagrams, and entity-relationship diagram. After making these diagrams, it can proceed with the specifications with RAISE methodology.

The results of the analysis phase is the initial specification of software that can be written by using the RSL. This initial specification is a requirement for software written in a human language. Final specification can be written also using RSL. Then the final specification will be translated into programs in a particular programming language at stage of translation.

In general, there are three approaches to distinguish the software specification techniques, namely:
• Applicative vs. imperative.
• Sequential vs. concurrent.
• Abstract vs. Concrete.

More complete descriptions of the RAISE methodology can be seen in The RAISE Specification Language, by The RAISE Language Group (1992).

RAISE specification language (RSL)

RSL is a language to make a formal software specification. RSL is used for the manufacture of a modular software specifications, each specification with the RSL, is made in modules that are interrelated to one another. Detailed descriptions of the RSL can be found at The RAISE Specification Language (RAISE Language Group, 1992).

In the RAISE tutorial (RAISE Method Group, 1995) and Chris George (George, 1998) presented that the so-called Implementation relations are relations between classes built on the specifications using RAISE. Implementation relation is used to determine the correctness on the development steps of a module, from a more abstract modules into a more concrete modules.

Class B implements class A, written as $A \preceq B$ if and only if:
• The signature of B includes the signature of A, and
• All properties of A hold in B.

CASE STUDY

The following will be described the registration in the academic information system IT Del, following the stages that exist in the business rules approach, i.e. Scope, Plan, Discover, and Analyze. While the stages of Design, and Deliver is not addressed in this paper because it is related to the implementation of the analysis results. In this paper the focus of attention at every stage of the business rules approach are the rules tracks, in accordance with the main focus of the paper.

Scoping

This stage aims to obtain an overview of existing systems. Based on existing documents, summarized the initial statement that explains what is included into the system and what is not included into the system. A scope statement includes the definition of objectives, the parties involved, business processes, location, subsystem, and data, which belong to the system to be built.
The purposes of the registration is as follows:
• Provide courses which opened to students.
• Facilitate the submission of a plan of study by students.
• Facilitate online consultation.
• Facilitate the finalization of the registration, carried out by officers of academic administration.

In the process of registration of students, actors involved are students, wali, and administrators. In addition to this process academic system also requires interaction with other systems, namely bursar for payment of the fee, and the library for the information of books borrowed by students, so that there are other actors involved, the bursar and librarian staff.

Plan

This stage is more focused to project managers in managing software development to be built. This paper does not discuss the details of this phase. But as an illustration, for the scope of software development, at this stage the expected deliverables include: organizational structure of the team, a team of policy documents, documentation of all procedures (communication, sharing of knowledge, changes), and the detailed schedule of the project.

Discover

This stage is the most important stages in the discovering rules; it aims to discover the rules that will be applied to systems built. By observing the use-case and use-case scenario, observed the scenario where the system make decisions based on certain criteria (rules). For the students proposed a plan of study, decisions and rules affecting the decision were presented in Table-1.

Table-1. Decicion and rule of student submit the study plan.

Decision	Rule	Rule number
Is login accepted?	Student ID must be element of the Student ID set	R1
	input password must be equal with the student password	R2
Do the course can be taken?	**If** the prerequisite has been taken **and** the course is available **then** the course can be taken.	R3
whether the total of credits is valid?	total credit = sum of course credit proposed by student.	R4
	If total credit > 0 **and** total credit < max total kredit **then** the total credit is valid	R5
	If GPA < 3.00 **then** max total	R6

	credit = 20	
	If GPA ≥ 3.00 **then** max total credit = 22	R7
	If total credit valid **then** the study plan is valid, **and** send notification to student's wali	R8

Rules that have been identified then classified, for R1 and R2 are the constraints, R3 and R5 is inference, R4 is computing, R6 and R7 is a guideline, while the R8 is the action-enabler.

From each of the rules that have been identified, further identified term and the fact of those rules. In accordance with the definition of the term is a noun or noun phrase agreed upon definition, for example of rule R3 the terms are the courses, curriculum and prerequisite. While the facts that can be revealed is that the student took the course, students follow a curriculum and courses have prerequisite requirements.

Further identification carried out following any decision to be taken by the system, until re-registration process is completed, the students registered for the course in that semester.

Analysis

Based on the identification of the rules, terms and facts, for student registration, the entity that has the potential are: Mahasiswa, Wali, Rencana studi, Mata kuliah, Kurikulum, Pembayaran, Penundaan, Bebas pinjam, and MK_Lulus. The entity relation between componets are shown in Figure-5.

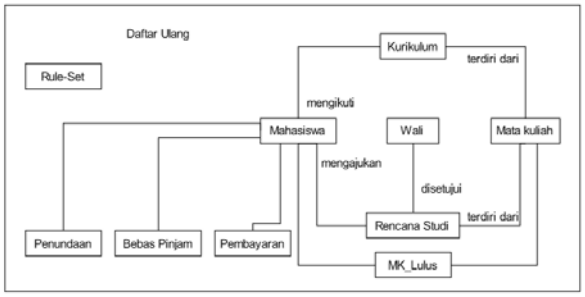


Figure-5. Entity relation, students registration.

The most important part of the business rules approach is that there is an entity Rule-sets are intended to keep all the rules that have been identified, separated from other components.

Rule enriched data model

Each rule associated with an entity are collected, the data model is modified to include attributes and any rules associated with each attribute, as shown in Figure-6.

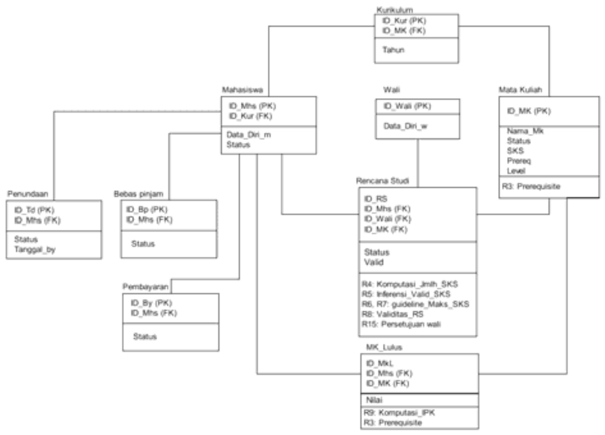


Figure-6. Rule enriched data model.

Rules analysis

Rules that have been identified made into atomic, namely that each rule represents only one thought, one way is to ensure that each rule may lead to an outcome.

As an examples, for R15: If the wali = agree, then status of study plan = approved and send email notifications to the student. R15 is an inference with two events which generate changes in study plans and delivery status notifications, so that it can be made into two rules:
R15a: if approval wali = agree, then study plan status = approved.
R15b: if approval wali = agree, then send approval email notifications to students.

Furthermore, the rules pattern are observed and interrelationships with other rules, and grouped according to attributes or entities that influence by the rules. The important thing is checking the completeness of the rule, if all possibilities of the premise has been defined the conclusions.

FORMAL SPECIFICATION

Based on the description of the entities on the submission of registration process of students and the rules involved, such as those modeled in Figure 5 and Figure 6, subsequently made a formal specification using the RAISE Specification Language, any entity created in one module, and given the name correspond with the name of the entity.

The most important part is the separation of the rules into a separate module. This is possible because the RSL is modular, so that every specification of an entity stored in a module, and can be referred to or called from other modules. Fragment of the module specification for concrete rules applicable at this stage are as shown in Figure-7.

VOL. 10, NO. 3, FEBRUARY 2015 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences
©2006-2015 Asian Research Publishing Network (ARPN). All rights reserved.
www.arpnjournals.com

```
scheme A_ATURAN1 =
  hide MHS, MK, MKL in
  with T1 in
    class
      object
        MHS: A_MAHASISWA1,
        MK: A_MATA_KULIAH1,
        MKL: A_MK_LULUS1,
    value
/* mata kuliah dapat diambil oleh mahasiswa */
    mk_can_chose_by_mhs =
      MK.ID_Mk × MHS.ID_Mhs → Bool
    mk_can_chose_by_mhs(idmk, idm) ≡
      ∃ idmkl: ID_MkL •
      prereq(idmk) ∈ list_mkl(tmkl(idmkl))
      ∧ status(tmk(idmk))
      pre idm = idm(tmkl(idmkl))

/* Menghitung jumlah dari nilai-dikali-sks nilai */
      sum_nilai: ID_MK-set → Nat
      sum_nilai(li) ≡
      if li = {} then 0 else
        let
          h = hd li,
          t = tl li,
          x = sum_nilai(t),
        in
          sks(MK.T_Mk(h)) *
          nilai_angka(nilai(MKL.T_MkL(h))) +
          sum_nilai (t)
        end
      end
    end
```

Figure-7. Specification of rule R1.

The rules associated with the provision to create, update, and delete (CRUD) on each entity remains pinned on these entities, while the rules between the entities is pulled out, the rules are stored on the module. Study Plan Specifications entities as shown in Figure-8.

```
  scheme A_RENCANA_STUDI1 =
    hide MHS, WL, MK, ATRN in with T1 in
      class
        object
          MHS: A_MAHASISWA1,
          WL: A_WALI1,
          MK: A_MATA_KULIAH1,
          ATRN: A_ATURAN1,
        type

    ID_Rs, No_urut: Nat0,
    List_Mk = MK.ID_Mk-set,
    RecRs ::
      idm: MHS.ID_Mhs ↔ chg_idm
      idw: WL.ID_Wali ↔ chg_tahun
      status: Status ↔ chg_status
      valid: Status ↔ chg_valid
      list_mk: List_Mk ↔ chg_list_mk,
    T_Rs = ID_Rs $\xrightarrow[m]{}$ RecRs,
  value
    /* generator */
    empty : T_Rs,
    empty≡ [ ],
    insert: ID_Rs× RecRs × T_Rs → T_Rs,
    insert(idrs, rs, trs) ≡ trs † [idrs ↦rs]
    remove: ID_Rs × T_Rs → T_Rs,
    remove(idrs, trs) ≡ trs \ {idrs},
    add_mk_to_rs: MK.ID_Mk × ID_Rs ×
    T_Rs → T_Rs,
    add_mk_to_rs(idmk, idrs, trs) ≡
      let
        rs = trs(idrs),
        new_rs =
        chg_list_mk(list_mk(rs) ∪ {idmk}, rs),
      in
        (trs † [idrs ↦ new_rs ])
      end
      pre ATRN.can_add_mk_to_rs (idmk, idrs, trs),
    remove_mk_rs: MK.ID_Mk × ID_Rs ×
      T_Rs → T_Rs,
    remove_mk_rs(idmk, idrs, trs) ≡
      let
        rs = trs(idrs),
        new_rs =
        chg_list_mk(list_mk(rs) \ {idmk}, rs)
      in
        (trs † [idrs ↦ new_rs ])
       pre defined_mk_rs (idmk, idrs),
    /* observer */
    defined : ID_Rs × T_Rs → Bool,
    defined(idrs, trs)≡ idrs ∈ dom trs,
    defined_mk_rs : MK.ID_Mk ×
      ID_Rs → Bool,
    defined_mk_rs(idmk, idrs) ≡
      idmk ∈ list_mk(idrs),
    lookup: ID_Rs × T_Rs $\overset{\sim}{\rightarrow}$ RecRs,
    lookup(idrs, trs) ≡ trs(idrs)
      pre defined(idrs, trs),
    lookup_status: ID_Rs × T_Rs $\overset{\sim}{\rightarrow}$ Status,
    is_valid_rs: ID_Rs → Bool,
    is_valid_rs(idrs) ≡jml_sks_rs(idrs) ≤
      ATRN.maks_sks(idrs),
    jml_sks_rs: ID_Rs $\overset{\sim}{\rightarrow}$ Nat
    jml_sks_rs(idrs) ≡
      ATRN.sum_sks(list_mk(trs(idrs))),
end
```

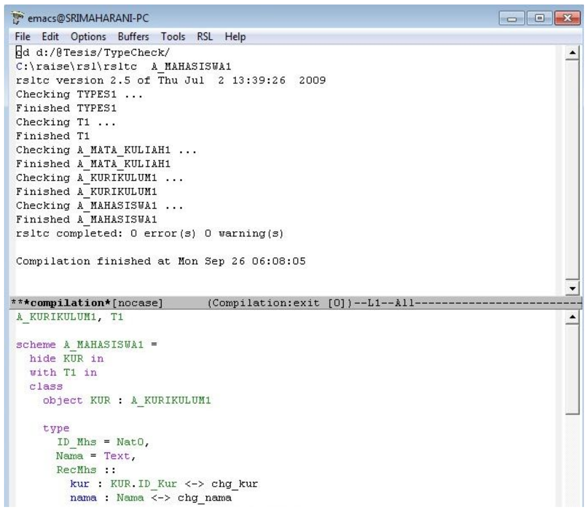
Figure-8. Study plan specification.

In a study plan specifications above, the module calls the rules associated with the addition of provisions subject to the plan of studies, validity rules, and rules of the credits.

FORMAL VERIFICATION

Verification of a formal specification of the application with business rules approach is done by checking whether the consistency maintained at each transition from one specification to the following specification in RSL specification stage. The checking uses the implementation relations. Checking assisted by using RSL Tools (rsltc).

Each module that models the entities of each component re-written with the notation that is appropriate for RSL Tools (rsltc). To check the consistency of the type of each module we do the type checking, as shown in Figure-9.



```
cd d:/@Tesis/TypeCheck/
C:\raise\rsl\rsltc  A_MAHASISWA1
rsltc version 2.5 of Thu Jul  2 13:39:26  2009
Checking TYPES1 ...
Finished TYPES1
Checking T1 ...
Finished T1
Checking A_MATA_KULIAH1 ...
Finished A_MATA_KULIAH1
Checking A_KURIKULUM1 ...
Finished A_KURIKULUM1
Checking A_MAHASISWA1 ...
Finished A_MAHASISWA1
rsltc completed: 0 error(s) 0 warning(s)

Compilation finished at Mon Sep 26 06:08:05

**compilation*[nocase]       (Compilation:exit [0])--L1--All------------------------
A_KURIKULUM1, T1

scheme A_MAHASISWA1 =
  hide KUR in
  with T1 in
  class
    object KUR : A_KURIKULUM1

    type
      ID_Mhs = Nat0,
      Nama = Text,
      RecMhs ::
        kur : KUR.ID_Kur <-> chg_kur
        nama : Nama <-> chg_nama
```

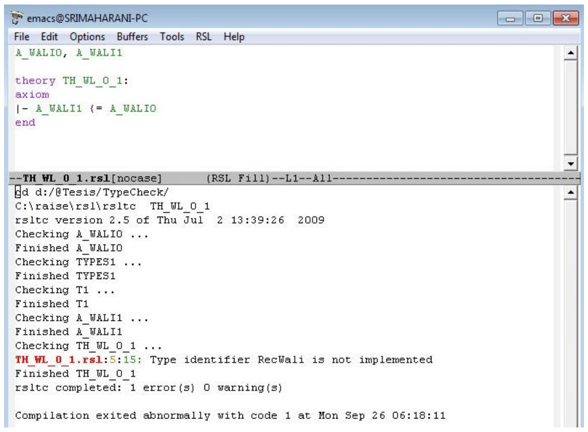
Figure-9. Type checking, students module.

Then for each transition from a module, we made a theory to verify the implementation relation. As an example for the transition from A_WALI1 A_WALI0 to be made as follows:

A_WALI0, A_WALI1
theory WL_0_1 :
  axiom
    $\vdash$ A_WALI1 $\preceq$ A_WALI0
  end

Figure-10. Module Wali implementation relation.

RSL tool will check whether the module A_WALI1 implemented module A_WALI0 in accordance with the implementaion relations criteria, the result of the checking the implementation relation of the above theory is shown in Figure-11.



```
A_WALI0, A_WALI1

theory TH_WL_0_1:
axiom
|- A_WALI1 {= A_WALI0
end

--TH_WL_0_1.rsl[nocase]       (RSL Fill)--L1--All-----------------------------------
cd d:/@Tesis/TypeCheck/
C:\raise\rsl\rsltc  TH_WL_0_1
rsltc version 2.5 of Thu Jul  2 13:39:26  2009
Checking A_WALI0 ...
Finished A_WALI0
Checking TYPES1 ...
Finished TYPES1
Checking T1 ...
Finished T1
Checking A_WALI1 ...
Finished A_WALI1
Checking TH_WL_0_1 ...
TH_WL_0_1.rsl:5:15: Type identifier RecWali is not implemented
Finished TH_WL_0_1
rsltc completed: 1 error(s) 0 warning(s)

Compilation exited abnormally with code 1 at Mon Sep 26 06:18:11
```

Figure-11. Implementation relation checking.

In the results above it appears there was one error which is that there is one identifier, RecWali, on A_WALI0 is not implemented in the A_WALI1. This is because the RecWali identifier in A_WALI0, is a sort while in A_WALI1, is a map from ID_Wali to Name. This can be corrected by changing RecWali in A_WALI0 into an abstract type.

CONCLUSIONS

Based on the study of the rules separation process on the construction of the system, which is examined through case studies the process of registration of students at the Del Institute of Technology, we concluded as follows:

Separation of rules on the system can be done by following the steps of software development with the approach of Business Rules Approach.

The rules that are identified from the system are separated and stored in a component, as an asset separate from other components in the system.

By using RSL, the rule component is represented as a rule module (rules scheme), as well as general representation in RSL modules.

Verification on the formal specification with rule systems as separate components has been done, and have met the criteria according to criteria of implementation relations, on the stage of software development using the RSL.

In further development, a few things to note is that the study has not define the aspect of concurrency, is associated with the assumption that the course participants have no limit minimum and maximum number of participants. If the restriction is entered into the system, it needs to be made concurrent version of the specification because of the availability of courses to be taken by students, the courses as a resource that is accessed by many students.

Also needs to be studied translation of research results into a form of specific programming language and do some test case, to complete the verification.

REFERENCES

Ross Ronald G. (2003), Principles of the Business Rule Approach, Addison Wesley, 2003.

Halle Barbara Von. (2002). Business Rules Applied, Building Better Systems Using the Business Rules Approach, Addison Wesley, 2002.

Cecilia Nugraheni. (2005). Diktat mata kuliah Metode Formal, 2005.

Liem Inggriani. (2003). Model Sistem Informasi Perguruan Tinggi, Penataran Pengembangan Sistem Informasi Manajemen Bagi Pimpinan dan Dosen Perguruan Tinggi Swasta di Lingkungan Kopertis Wilayah IV Bandung, 2003.

The RAISE Language Group (1992). The RAISE Specification Language, Prentice Hall, 1992.

The RAISE Method Group (1995). The RAISE Development Method, Prentice Hall International (UK) Limited, 1995.

George Chris. (1998). A RAISE Tutorial, The United Nation University UNU/IIST, 1998.