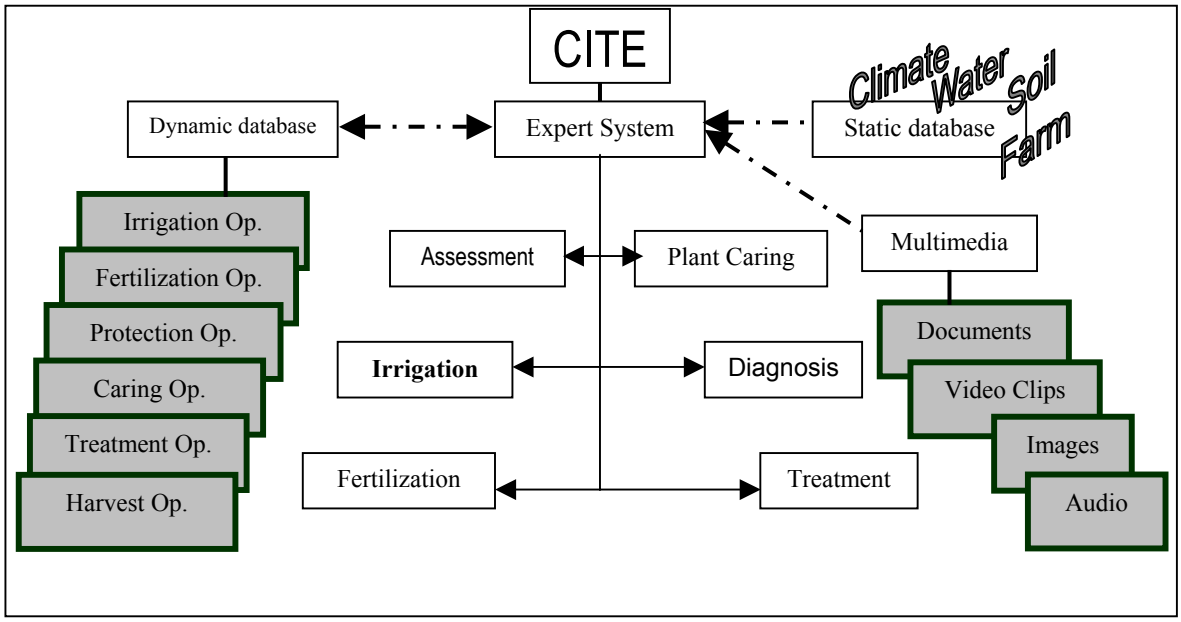
1. Introduction

The main goal of developing the expert system for Citrus (Orange and Lime) production version four is to integrate two versions of expert systems for orange and lime and to include the multimedia system to be integrated with the expert system. Consequently this new version is based on earlier versions of CITEX and LIMEX expert systems (TR/CLAES/11/97.5) titled with "Detailed Design of the Citrus Production Management in Field Agriculture of Citrus Fruits (CITEX) version 2.2" and (TR/CLAES/CRG[11]/8) titled with "Detailed design of the expert system for LIMEX version 1.0". The requirement specification (TR/CLAES/41/98.9) for CITEX4 has been considered. According to the requirement specification CITEX4 is divided into three subsystems, each of which is concerned with solving a specific problem: the Expert System (ES), Multimedia Information System (MIS), and farm database system (see figure 1). The ES and MIS are integrated together to give better enhancement for CITEX users.

Figure 1 illustration of CITEX integrated system



The system introduced in this report has been influenced by the remarks obtained from the verification reports for CITEX version 2.2 (TR/CLAES/32/98.4) and (TR/CLAES/39/98.8) and obtained also from evaluation report for LIMEX version 1.0 (TR/CLAES/18/97.10). Other important source for this design report our domain expert, to review the treatment materials used to remedy the disorder, which the canceled materials were removed from the design and replaced with the new materials. The last source for this report is the multimedia report for CITEX4 (TR/CLAES/53/99.1) and the database report for CITEX4 (TR/CLAES/56/99.2).

The current version of this system includes the following changes:

- Integrate CITEX (Orange) and LIMEX (Lime) expert systems in one package.
- Include multimedia information system and integrate this part with the expert system to support each subsystem with the suitable media.
- Include database for recording all events for the horticulture during the year.
- Port the new system to run under Arabic Windows 95, 98 and NT.

In this report we will focus on the irrigation subsystem which is a one component of the expert system part of **CITEX4** as shown in figure 1.

In the next section the components of the irrigation system is presented. The domain layer is described in section three. In section four the inference layer is presented. Then the task layer is described in section five. The user interface screens described in section six. The suggested multimedia to be integrated in this part of the expert system is presented in section seven. Finally there are some test cases generated form this design presented in section eight.

2. Irrigation components

Optimum management of irrigation schedule requires consideration of an extensive number of complex variables, relative to the soil, water, climate, plant, horticulture, agriculture operations, organic manure, and disorder existence. The main goal of this function is to obtain the optimal quantity of water taking into consideration all these variables.

This design based on the report titled with "Generic Irrigation Design Applied in Tomato Crop" (TR/CLAES/72/99.5) because this generic design covers all aspects of any irrigation design for specific crop with some modification. Consequently the irrigation design for **citex4** (Orange and Lime) is based on the main features of the generic design for irrigation in open field part only.

The objective of developing this system is to provide the user the suitable irrigation schedule for his horticulture of citrus (Orange and Lime) according to the characteristics of this horticulture.
The design provided here covers the following issues:-
Irrigation schedule could be obtained either by daily or monthly bases. The first depends on the availability of weather station for providing the daily climate input parameters. Where, the second depends on the availability of climate reference data.
The farm type is open field. Therefore, only one method of Et0 calculations (Pennman) has been considered.
The farms might have different irrigation systems including flooding, drip, or sprinkler.

This system integrates with the multimedia system, so the user can retrieve relevant media from the expert system output or during consultation from the already stored media in the multimedia system.

The output of this system is the irrigation schedule for the orange or lime crop within a period determined by the user. This schedule includes the water quantity, irrigation frequency, and the time required to operating the pump for each month in addition to the fasting period for lime. The output should include the following parameters:

Concept	Property	Legality Checks
Irrigation Operation	Irrigation Date /Month	Date / >= 1 <=12
	Irrigation Interval	>= 1 <= 30
	Irrigated Water Qty/Faddan	>= 1 <= 4200 M^3
	Motor Pump Time /Day	>= 0 <= 10

3. Irrigation Problem Description

Irrigation scheduling in the agriculture domain is concerned with the water quantity for each time instance. Figure 2 illustrates the inputs and the output of the irrigation for open field.

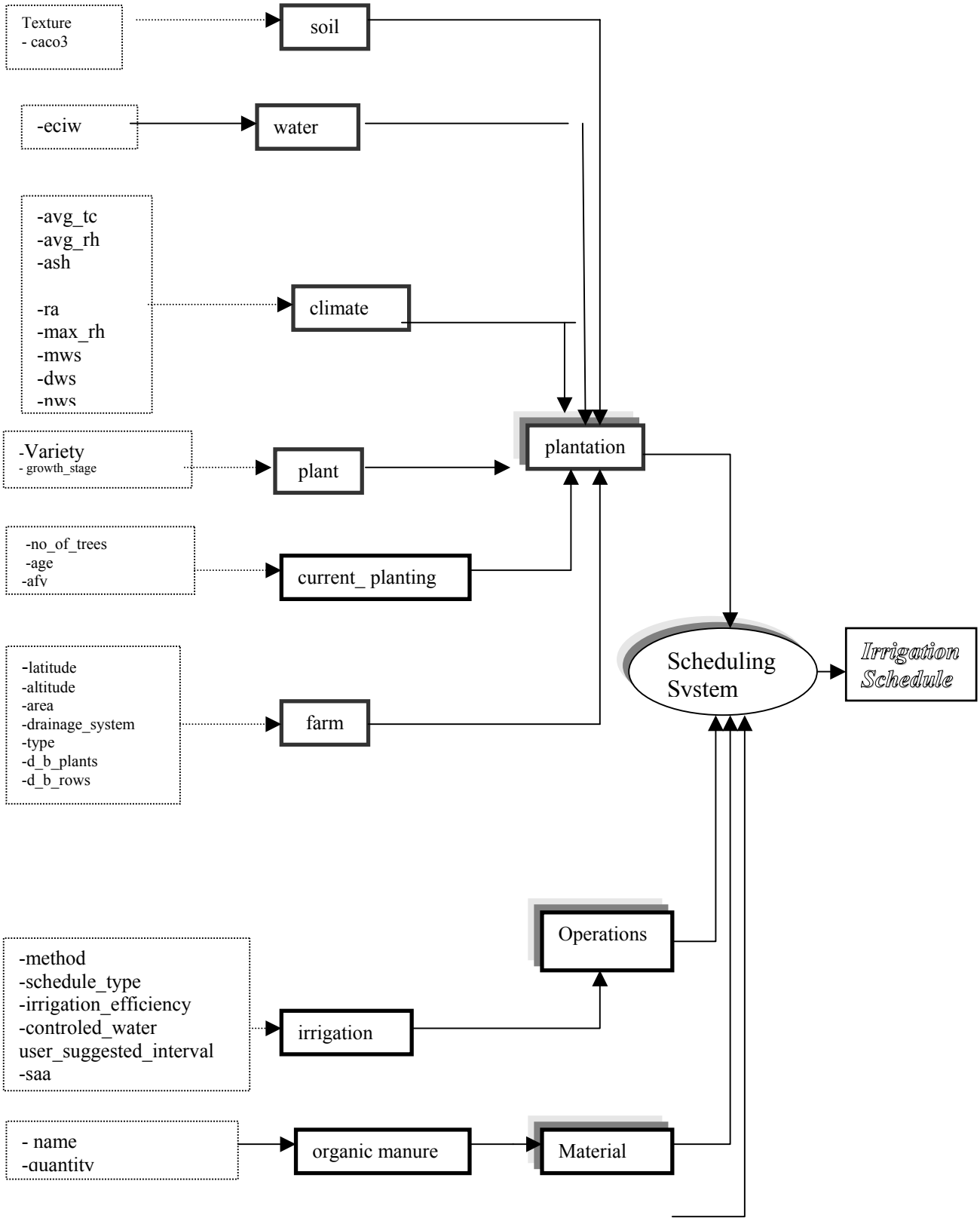


Figure 2: A Conceptual View of the irrigation Function

4. Irrigation System Architecture

The overall design has five main components: *control knowledge*, *meta knowledge & mechanism selector*, *domain knowledge, repository and case models*. The control knowledge consists of three basic parts *task knowledge*, *inference knowledge* and *library of primitive problem solvers*. Figure 2 shows a complete system architecture for irrigation scheduling problem.

- The control knowledge consists of the task, inference steps and a set of mechanisms. The task controls the inference steps sequence of actions that should be done in order to solve the problem under consideration. The inference steps evaluate, select and invoke the appropriate mechanism from the library to achieve the inference goal.

  The task control knowledge consists of three basic parts task knowledge, inference knowledge and library of primitive problem solver. The task knowledge composed of three sub-tasks namely: "*evaluate*" , "*propose*" and "revise" and two transfer task are representing the interaction with user. All primitive sub-task mapped into a set of inference steps, for instance, the "evaluate" sub-task is mapped to "assess" and "recommend" inference steps.

  Each mechanism could be used by one or more inference step. For instance, the implication mechanism is used by "*assess*" and "*recommend* (since the domain models used by there inferences are represented using the implication rules).

- The meta knowledge is the declarative part of the control knowledge component in our architecture. It is used for the selection of the appropriate mechanism to be executed through the inference steps. This knowledge includes: mechanism type, and domain knowledge representation that a mechanism operates on. The mechanism selector is a module serves as interface between the inferences and the library of problem solvers for selecting the suitable mechanism. It interacts with the meta knowledge to perform its job.

- The domain knowledge layer consists of two parts: the set of *ontology* related to the irrigation schedule and *domain models*. The class of ontology includes computational,

plantation, operations, plantation_factors, material, and disorder ontology. The set of domain models includes evaluation, association, expansion, computational, and revise. The domain model can be used one mechanism and one or more ontology. For instance, the computational model used mathematical dependency mechanism and computational, plantation and operations ontology.

- This integration is done through special data structure called *repository*, which receives output resulted form each problem solver during its execution.

- The case model is the dynamic data base part of the system architecture. It is used for store all events related to the irrigation problem.

5. Domain Knowledge

The domain knowledge consists of two parts: the set of *ontology* related to the application domain and *domain models*. Figure 3 shows a complete domain knowledge schema for irrigation scheduling problem.

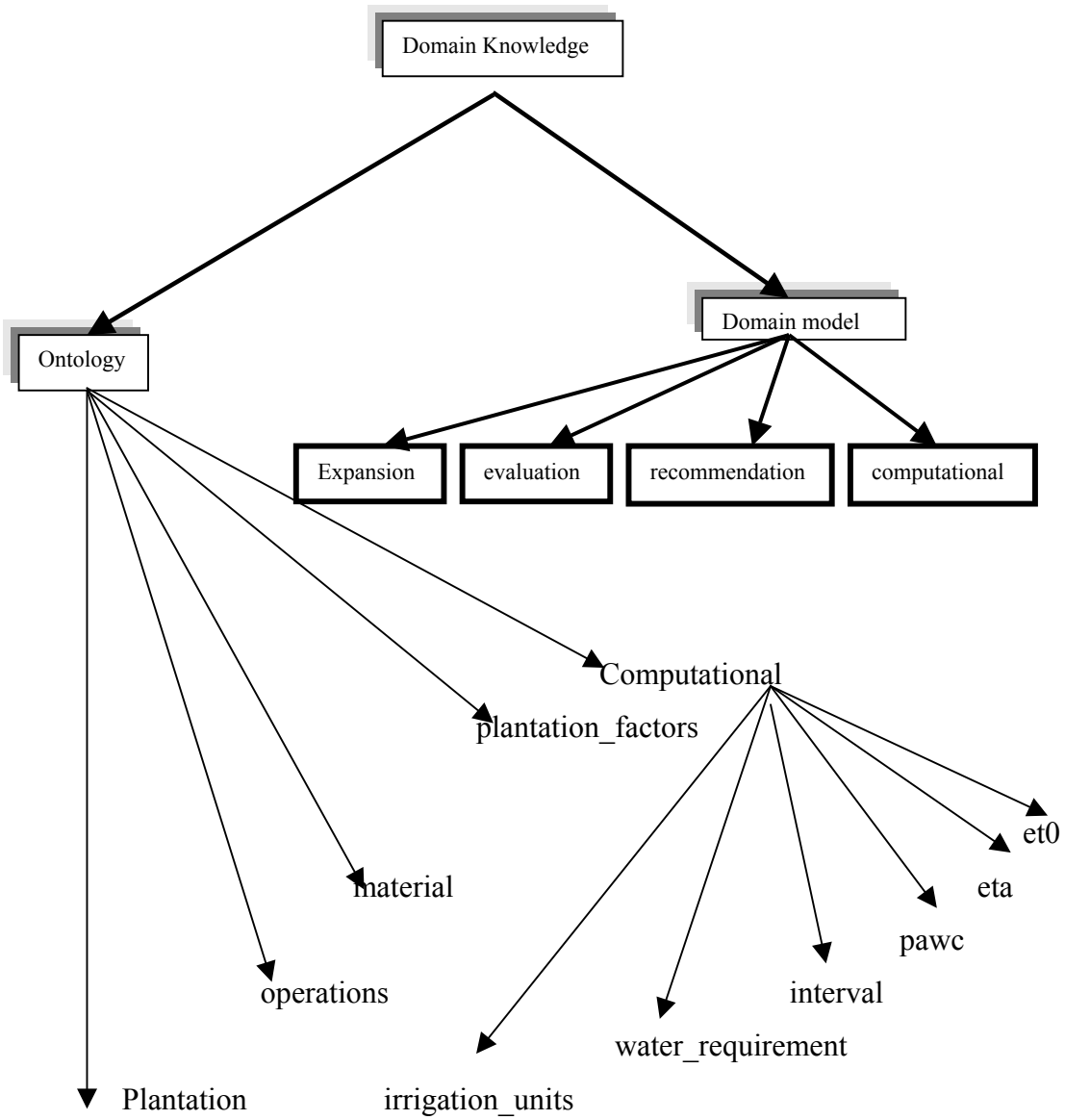


Figure 3: Domain Knowledge for irrigation schedule Problem

Ontology

The “ontology” can be defined as *"what exist in the real life application"*. It can be viewed also as an explicit specification of a conceptualization: the objects, concepts, and other entities that are assumed to exist in some area of interest and the relationships that hold among them [Gruber T.R., 1992]. These part has been described in the two points of view namely: conceptual view and ontology specification.

4.1.1 Conceptual View

The conceptual view present the overall description about the ontology as shown in figure 3. The ontology uses in these application consisting of a set of concepts namely: plantation, operations, material, plantation factor, et0, eta, pawc, interval, water_requirement, and irrigation_units. These concepts are shown in figures 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14 and 15 respectively.

Each ontology contains a set of related concepts forming a hierarchy structure. A complete description related to these hierarchy is represented as a graph containing nodes representing concepts and arcs representing the relationships between these concepts..

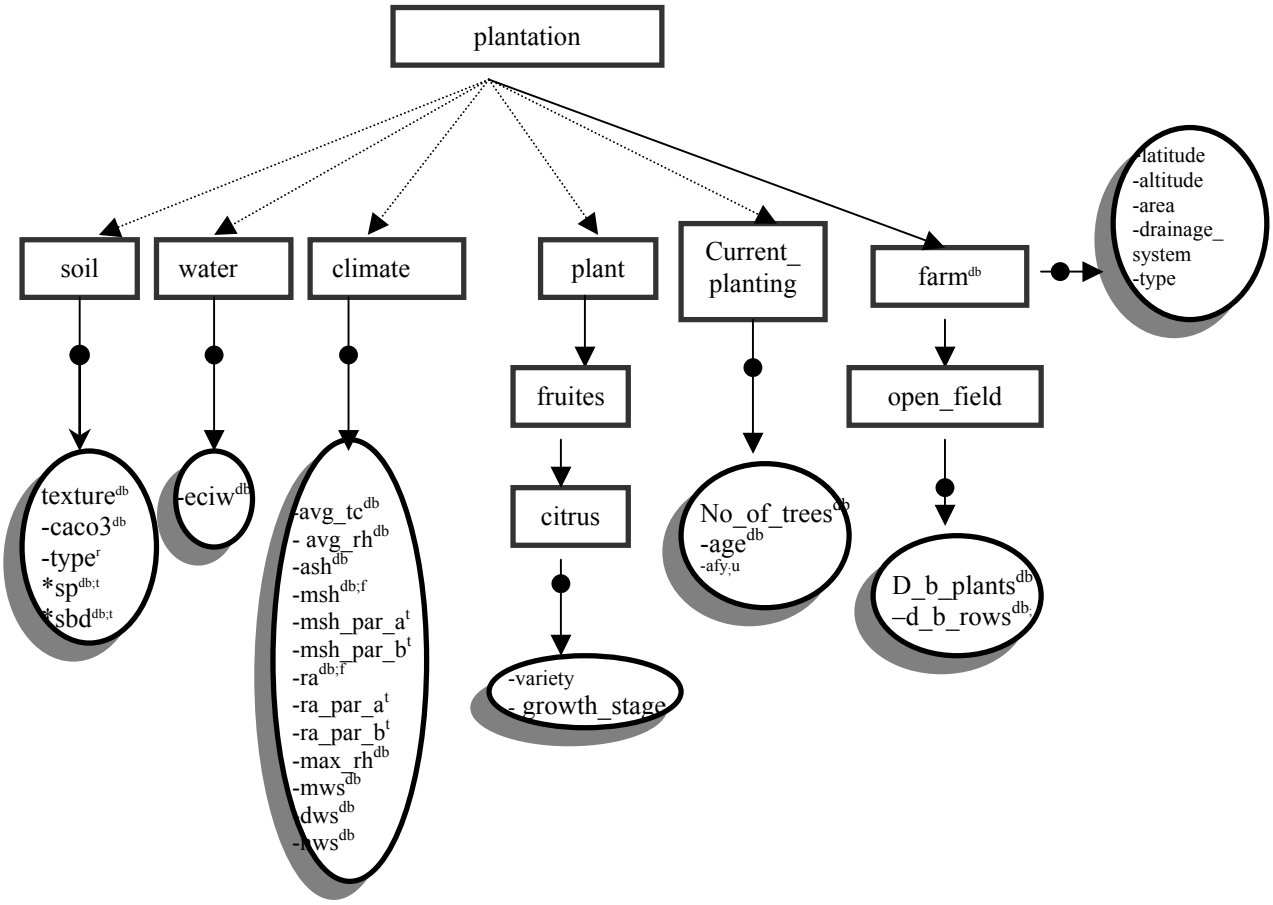


Figure 4: A Conceptual Overview of Plantation

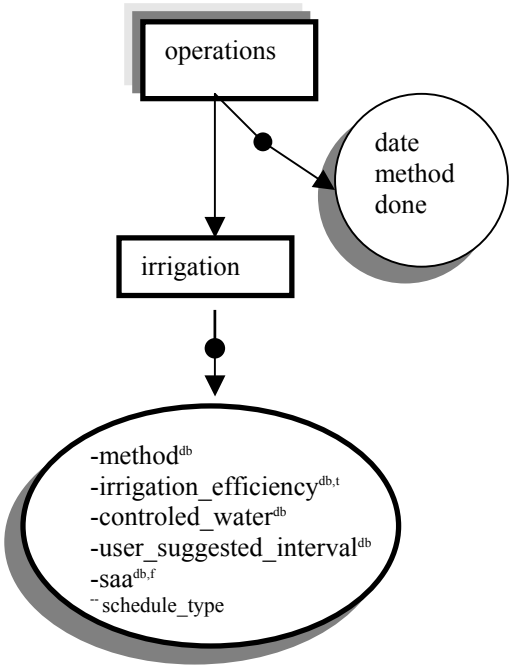


Figure 5: A Conceptual Overview of Operations

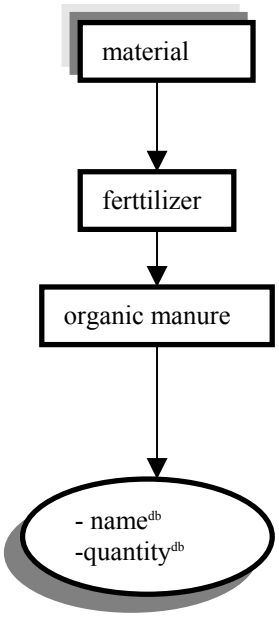


Figure 6: A Conceptual Overview of Material

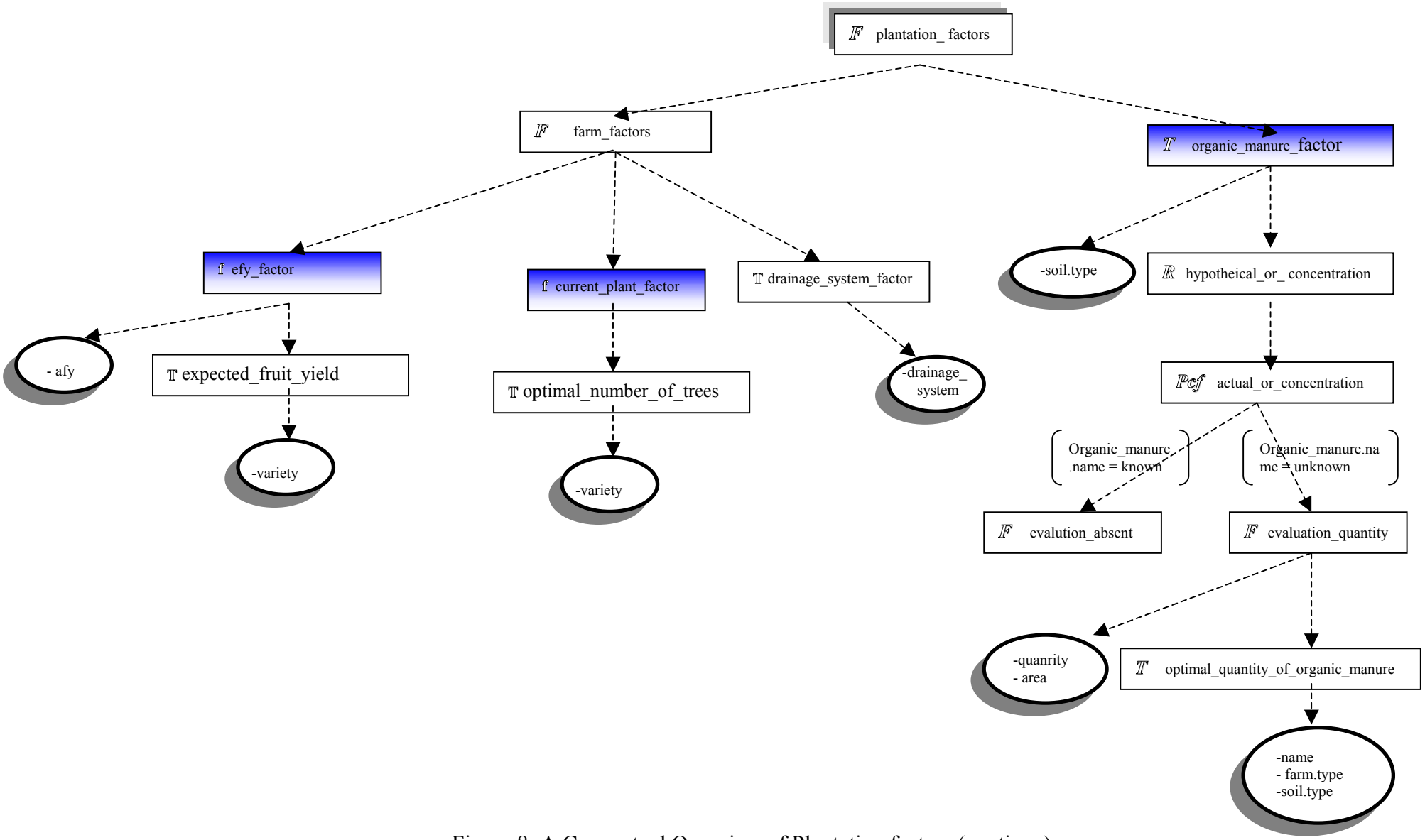


Figure 8: A Conceptual Overview of Plantation factors (continue)

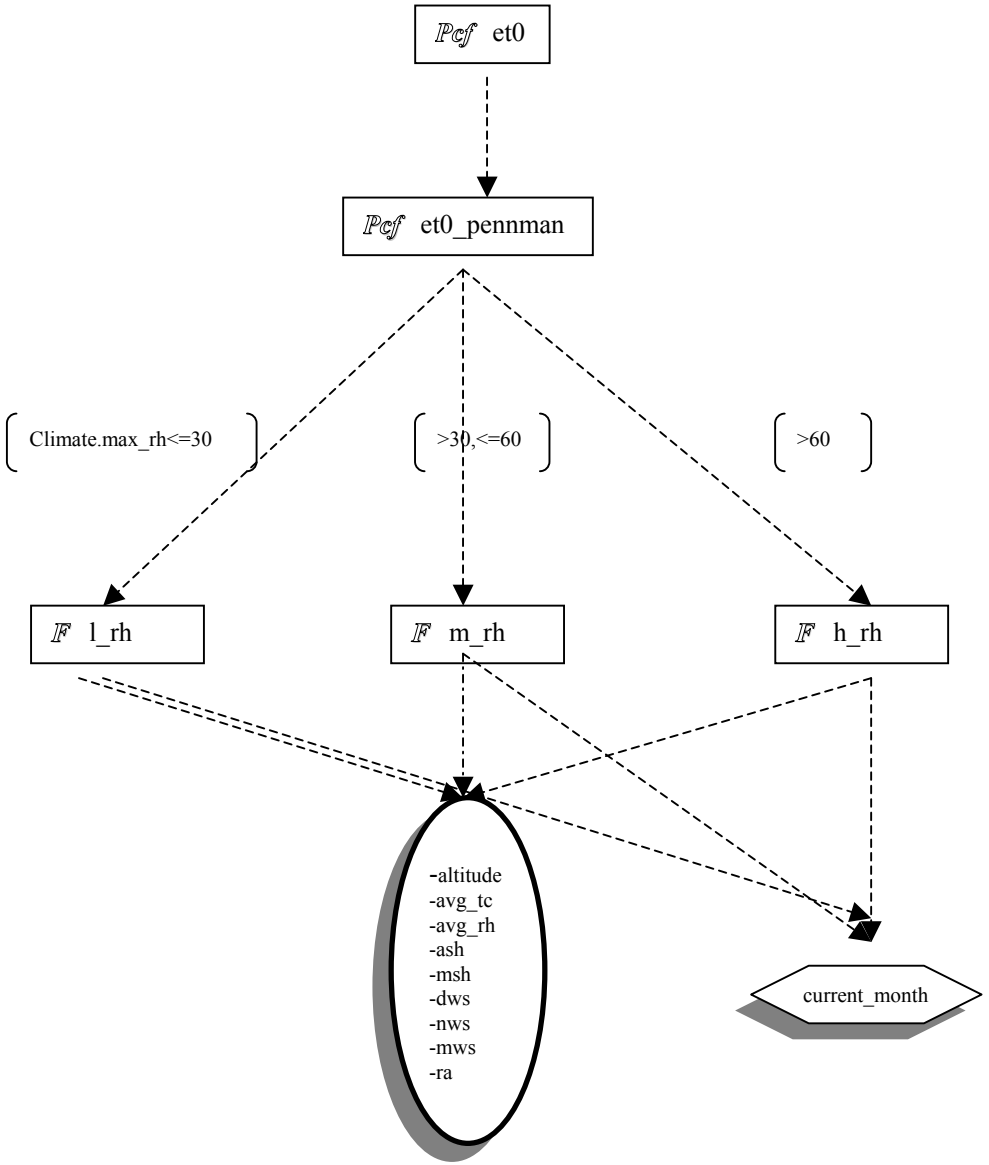


Figure 9: A Conceptual Overview of Et0

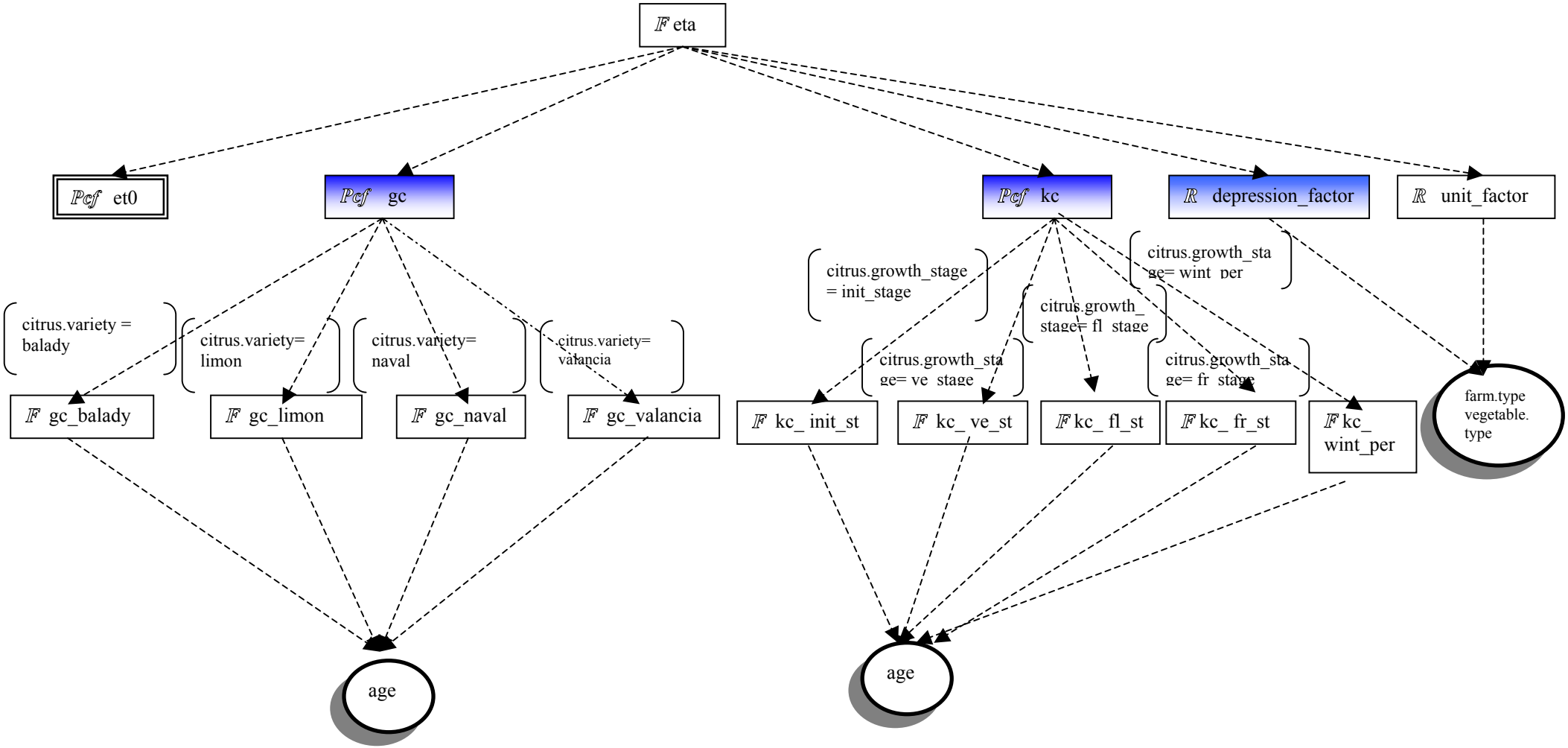


Figure 10: A Conceptual Overview of Eta

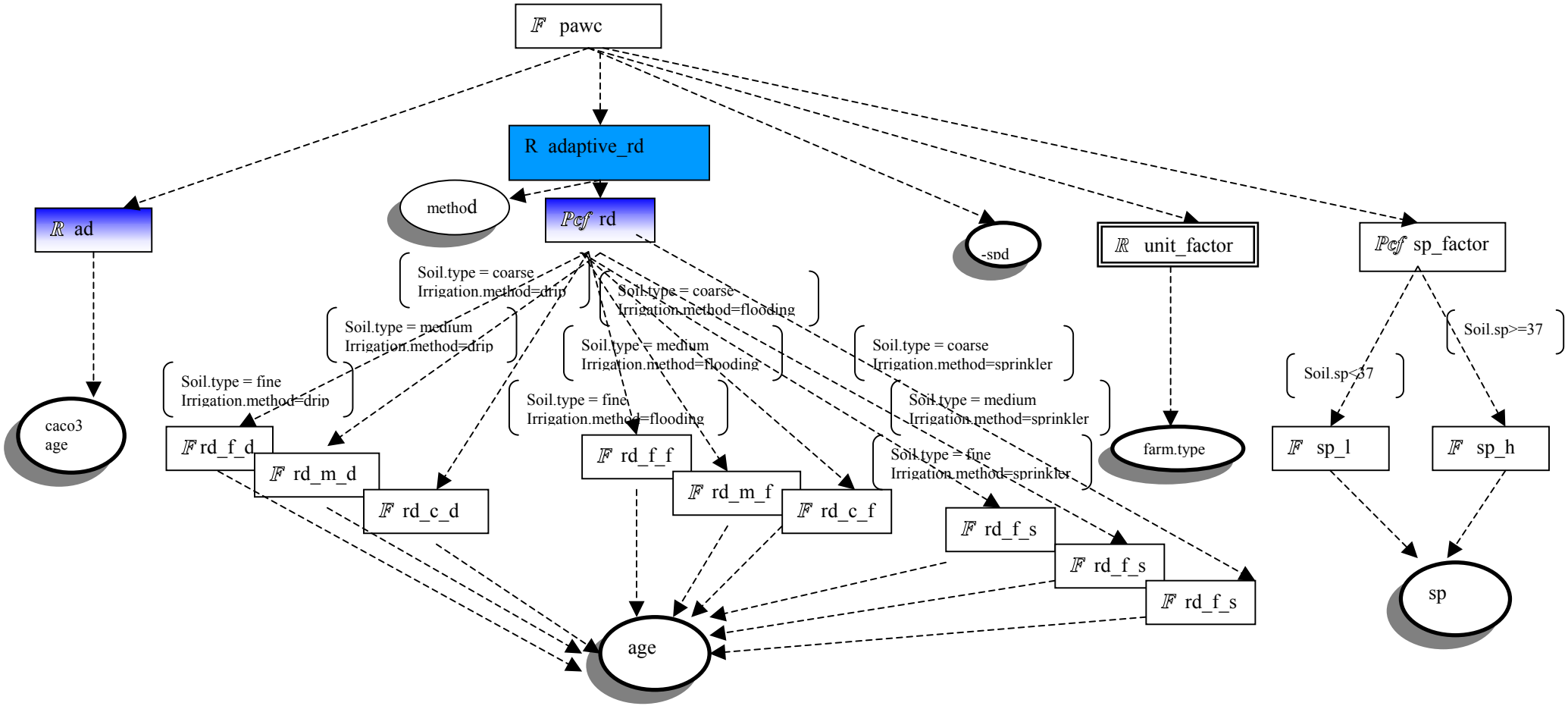


Figure 11: A Conceptual Overview of Pawc

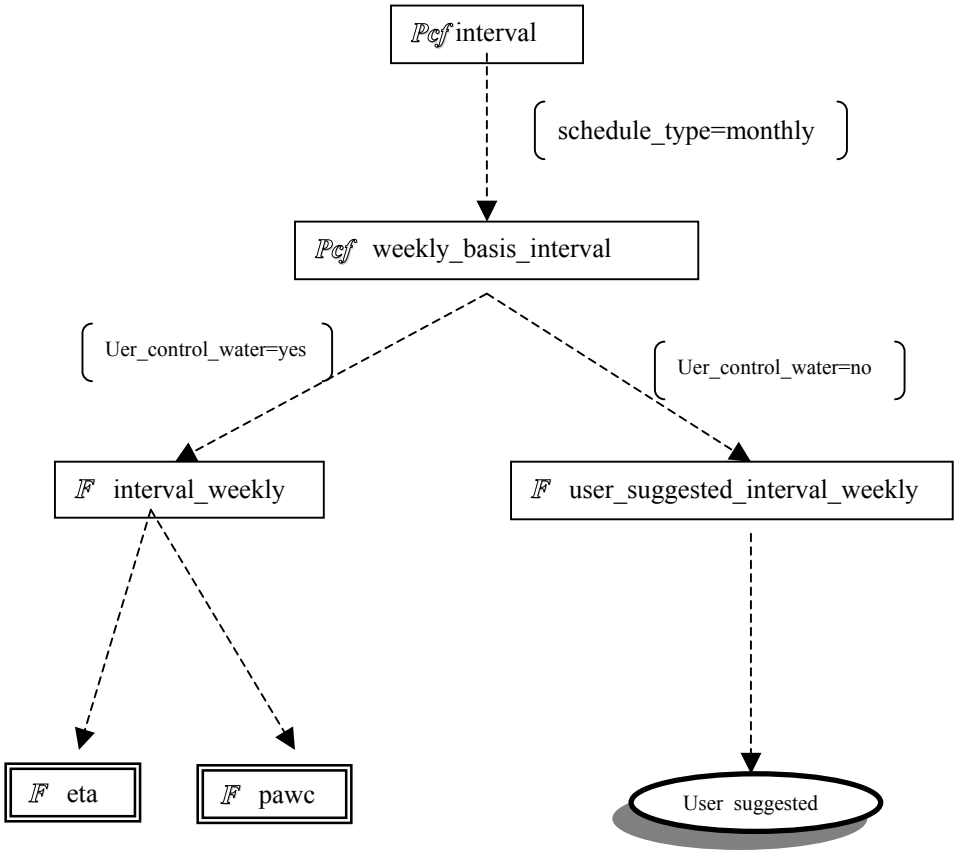
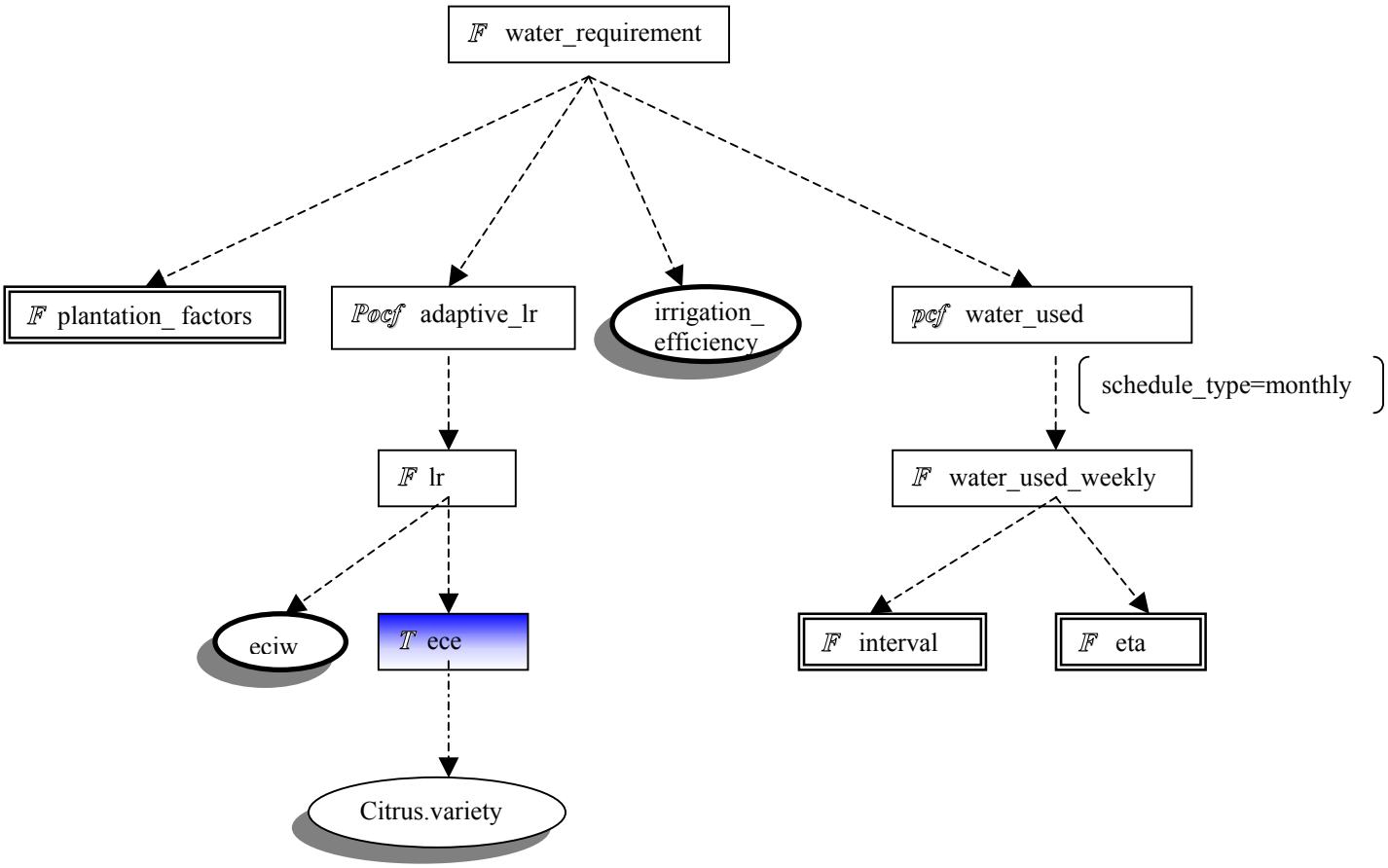


Figure 12: A Conceptual Overview of interval


water_requirement
plantation_ factors
adaptive_lr
irrigation_ efficiency
water_used
schedule_type=monthly
lr
water_used_weekly
eciw
ece
interval
eta
Citrus.variety

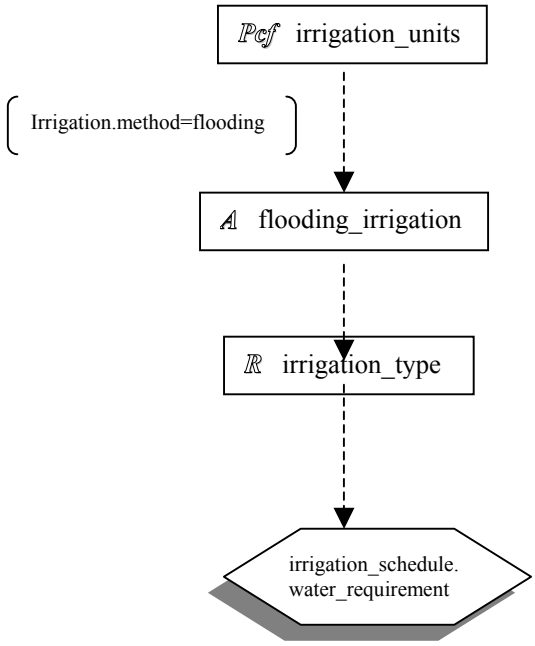


Figure 15: A Conceptual Overview of irrigation_units

Ontology Specification

ONTOLOGY plantation_ontology;
TYPE general;

6.1.1.1 Plantation Ontology

One) Concept

Concept	Description
Plantation	concept plantation;
Soil	concept soil; properties : texture: NOMINAL, VALUE-LIST(clay; clay loam; silt clay; silty clay loam; sandy clay; sandy clay loam; silt loam; silty loam; sandy loam; sand; loamy sand"), SOURCE(D.B.) SINGLE. Caco3 : REAL, NUMBER-RANGE(0,1000), SOURCE(D.B) SINGLE. Nessary. type: NOMINAL, VALUE-LIST(fine,medium,coarse), SOURCE(Derived[relation,soil_type]), SINGLE sp : REAL, %soil saturated percentage NUMBER-RANGE(0,1000), SOURCE(D.B.; Derived[table, sp_t]), SINGLE. sbd: REAL, %soil bulk density NUMBER-RANGE(0,1000), SOURCE(D.B.; Derived[table,sbd_t]) SINGLE.
water	concept water; properties : eciw : REAL, NUMBER-RANGE(0,3), SOURCE(D.B.) SINGLE.
climate	concept climate; properties : avg_tc : REAL,% average temperature NUMBER-RANGE(0,40), SOURCE(D.B.) MULTIPLE. avg_rh : REAL, %average relative humidity NUMBER-RANGE(0,100),

Concept	Description
	SOURCE(D.B.) MULTIPLE. ash : REAL, % actual sun shine hours NUMBER-RANGE(5,17), SOURCE(D.B.) MULTIPLE. msh: REAL, % maximum sun shine hours NUMBER-RANGE(5,17), SOURCE(D.B.; DERIVED[function,msh_f]) MULTIPLE. ra: REAL, %extra radiation NUMBER-RANGE(3,17), SOURCE(D.B.; DERIVED[function,ra_f]) MULTIPLE. max_rh : REAL, %maximum relative humidity NUMBER-RANGE(0,100), SOURCE(D.B.) MULTIPLE. mws : REAL %mean wind speed NUMBER-RANGE(0,100) SOURCE(D.B.) MULTIPLE. dws: REAL %day wind speed NUMBER-RANGE(0,100) SOURCE(D.B.) MULTIPLE. nws: REAL %night wind speed NUMBER-RANGE(0,100) SOURCE(D.B.) MULTIPLE. msh_par_a: REAL, % maximum sun shine hours % parameter (a). NUMBER-RANGE(-10,17), SOURCE(DERIVED[table,msh_t]) SINGLE. msh_par_b: REAL, % maximum sun shine hours % parameter (b). NUMBER-RANGE(-10,17), SOURCE(DERIVED[table,msh_t]) SINGLE. ra_par_a: REAL, %solar radiation parameter (a) NUMBER-RANGE(-10,17), SOURCE(DERIVED[table,ra_t]) SINGLE. ra_par_b: REAL,% solar radiation parameter(b) NUMBER-RANGE(-10,17),

Concept	Description
	SOURCE(DERIVED[table,ra_t]) SINGLE.
Plant	concept plant.
fruites	concept fruites; sub-type-of : plant.
citrus	concept citrus; sub-type-of : fruites. Properties : variety : NOMINAL VALUE_LIST((balady, naval, valancia, limon)) SOURCE(DataBase) SINGLE. growth_stage : NOMINAL VALUE_LIST((init_stage, wint_per, ve_stage, fl_stage, fr_stage)) SOURCE(DERIVED[table, growth_stage_t]) SINGLE.
farm	concept farm; properties : latitude: REAL NUMBER-RANGE(0,1000) SOURCE(D.B.) SINGLE. altitude : REAL NUMBER-RANGE(0,1000) SOURCE(D.B.) SINGLE. area: REAL, NUMBER-RANGE(1,2000), SOURCE(D.B.) SINGLE drainage_system: NOMINAL, VALUE-LIST(good;medium;bad), SOURCE(D.B.) SINGLE type: NOMINAL VALUE_LIST([open_field]) SOURCE(D.B.) SINGLE. NECESSARY.
open_field	Concept open_field; sub-type-of : farm. properties : d_b_plants: REAL NUMBER_RANGE(0,1000) SOURCE(D.B.) SINGLE. NECESSARY. d_b_rows: REAL NUMBER_RANGE(0,1000) SOURCE(D.B.) SINGLE.

Concept	Description
	NECESSARY.
current_planting	Concept current_planting; properties : no_of_trees: INTEGER, NUMBER-RANGE(1,2000), SOURCE(D.B.) SINGLE NECESSARY. age: REAL NUMBER_RANGE(0,1000) SOURCE(D.B.) SINGLE. NECESSARY. Afy : REAL, NUMBER-RANGE(1,2000), SOURCE(USER) SINGLE NECESSARY.

b)Relation

Relation	Description
Soil_type	relation soil_type; properties: texture, type; argument-1: soil; argument-2: soil; axioms : ***R1: If*** (texture of soil = "clay; clay loam; silty clay; silty clay loam" ***Then*** type of soil = fine ***R2: If*** (texture of soil = "sandy clay; sandy clay loam; silt loam; silty loam" ***Then*** type of soil = medium ***R3: If*** (texture of soil = "sandy loam; sand; loamy sand" ***Then*** type of soil = coarse

Two) Table

Sp_t

table sp_t; %soil saturated percentage
Input([soil.type])
Output([soil.sp]).

Input		output	
Concept	Property	Concept	Property
Soil	type	Soil	Sp
	fine		25
	medium		22
	coarse		15

sbd_t **table** sbd_t %soil bulk density;
Input([soil.type])
Output([soil.spd]).

Input		**output**	
Concept	Property	Concept	Property
Soil	type	Soil	Spd
	fine		1.3
	medium		1.35
	coarse		1.4

ra_t **table** ra_t;
Input([task_parameters.current_month])
Output([climate.ra_par_a,climate_ra_par_b]).

input		output			
task_parameters	**Current_month**	**climate**	Ra_par_a	**climate**	**Ra_par_b**
	1		15.26147		- 0.21331
	2		15.89913		- 0.17519
	3		16.18918		- 0.10636
	4		15.76883		- 0.02416
	5		14.708		0.0529
	6		14.708		0.0529
	7		14.708		0.0529
	8		14.708		0.0529
	9		15.84288		- 0.06929
	10		15.95455		0.14701
	11		15.49091		0.20383
	12		15.12208		0.23039

msh_t **table** msh_t;
Input([task_parameters.current_month])
Output([climate.msh_par_a,climate.msh_par_b]).

Input		output			
Task_parameters	**current_month**	**climate**	**msh_par_a**	**climate**	**msh_par_b**
	1		12.0989		- 0.0455
	2		12.0989		- 0.0455
	3		12.01444		- 0.00233
	4		12.00222		0.031

	5		11.97556		0.057333
	6		11.9467		0.06933
	7		11.9467		0.06933
	8		12.00667		0.041333
	9		12.01556		- 0.012
	10		12.06222		- 0.02033
	11		12.1322		- 0.0588
	12		12.1322		- 0.0588

growth_stage_t **table** growth_stage_t;
Input([task_parameters.month, cirtus.variety)
Output([citrus.growth_stage]).

Input				**Output**	
Task_parameters	**month**	**cirtus**	**variety**	**citrus**	**growth_stage**
	1		balady		wint_per
	2		balady		init_stage
	3		balady		ve_stage
	4		balady		fl_stage
	5		balady		fr_stage
	6		balady		fr_stage
	7		balady		fr_stage
	8		balady		fr_stage
	9		balady		fr_stage
	10		balady		wint_per
	11		balady		wint_per
	12		balady		wint_per
	1		naval		wint_per
	2		naval		Ve_stage
	3		naval		fl_stage
	4		naval		fr_stage
	5		naval		fr_stage
	6		naval		fr_stage
	7		naval		fr_stage
	8		naval		fr_stage
	9		naval		fr_stage
	10		naval		init_stage

	11		naval		wint_per
	12		naval		wint_per
	1		valancia		wint_per
	2		valancia		fr_stage
	3		valancia		fr_stage
	4		valancia		fr_stage
	5		valancia		fr_stage
	6		valancia		fr_stage
	7		valancia		ve_stage
	8		valancia		ve_stage
	9		valancia		init_stage
	10		valancia		fl_stage
	11		valancia		wint_per
	12		valancia		wint_per
	1		limon		wint_per
	Input			**Output**	
Task_parameters	**month**	**cirtus**	**variety**	**citrus**	**growth_stage**
	2		limon		ve_stage
	3		limon		fl_stage
	4		limon		fr_stage
	5		limon		fr_stage
	6		limon		fr_stage
	7		limon		fr_stage
	8		limon		init_stage
	9		limon		wint_per
	10		limon		wint_per
	11		limon		wint_per
	12		limon		wint_per

d) Function

Function	Description
ra_f	Climate.ra = climate.ra_par_a + climate.ra_par_b* farm.latitude
msh_f	Climate.msh = climate.msh_par_a + climate.msh_par_b* farm.latitude

6.1.1.2 Operations Ontology

ONTOLOGY operations_ontology;
TYPE general;

a) Concept

Concept	Description
Operations	concept operations; properties : date: DATE SINGLE. method: NOMINAL MULTIPLE. done: NOMINAL SINGLE.
irrigation	concept irrigation; sub-type-of : operations. properties : method: NOMINAL, VALUE-LIST(drip; flooding; sprinkler) SOURCE(D.B.) SINGLE NECESSARY. schedule_type : NOMINAL, VALUE-LIST(monthly), SOURCE(User). irrigation_efficiency: INTEGER, NUMBER_RANGE(0;100), SOURCE(D.B., derived[table, irrigation_efficiency_t]) SINGLE. controled_water: NOMINAL, VALUE-LIST(yes;no), SOURCE(D.B.) SINGLE NECESSARY. user_suggested_interval: integer, NUMBER-RANGE(1;30), SOURCE(D.B.) SINGLE. saa: REAL, %soil absorbed area NUMBER-RANGE(0,1000), SOURCE(D.B.; Derive[table, Saa_t]) SINGLE.

b) Table

irrigation_efficiency_t

table irrigation_efficiency_t;
Input([irrigation.method])
Output([irrigation. Irrigation_efficiency]).

Input		**output**	
Concept	Property	Concept	Property
irrigation	method	irrigation	Irrigation_efficiency
	Drip		0.9
	flooding		0.75
	spinkler		0.6

saa_t

table saa_t;
Input([irrigation.method,soil.type])
Output([irrigation.saa]).

	input		**output**		
Concept	property	concept	Property	Concept	Property
irrigation	method	soil	type	irrigation	saa
	flooding		fine		80
	flooding		medium		70
	flooding		coarse		60
	drip		fine		25
	drip		medium		25
	drip		coarse		25
	sprinkler		fine		25
	sprinkler		medium		25
	sprinkler		coarse		25

6.1.1.3 Material Ontology

ONTOLOGY material_ontology;
TYPE general;

Concept	Description
fertlizer	concept fertlizer; sub-type-of : material
material	concept material.
organic_manure	concept organic_manure; sub-type-of : fertilizer. properties : name: NOMINAL, VALUE-LIST(cattle, hog, chicken, chicken bayade, chicken magazer, sheep, hourse, pigeon), SOURCE(D.B.) SINGLE

Concept	Description
	quantity : REAL, NUMBER-RANGE(0,100), SOURCE(D.B.).

6.1.1.4 Plantation_factors Ontology

a) Concept

Concepts are used to represent classes of objects in the irrigation domain. Every concept has a name, possible super concepts, a number of properties and a set of axioms. The axioms include node description. Each node is described in two parts: type of node and the source of value. There are six node types of relation, table, function, pre-conditions function, post conditions function and aggregation.

Concept	Description
plantation_factors	Concept Plantation_factors; sub_type_of: adjust_factor; Properties: visited: NOMINAL, VALUE-LIST(yes;no), SOURCE(Derived), SINGLE¡ value: real, NUMBER-RANGE(0;1000), SOURCE(DERIVED) SINGLE. axioms: node(function , Plantation_factors_f, _)
farm_factors	Concept farm_factors; sub_type_of: plantation_ factors; axioms: node(function , farm_factors_f)
efy_factor	Concept efy_factor; sub_type_of: farm_factors; axioms: node(function, efy_factor_f)
Expected_fruit_yield	Concept Expected_fruit_yield %efy sub_type_of: efy_factor; axioms: node(table, expected_fruit_yield_t)
current_plant_factor	Concept current_plant_factor; sub_type_of: farm_factors; axioms: node(function, current_plant_factor_r)
optimal_number_of_trees	Concept optimal_number_of_trees sub_type_of: current_plant_factor;

Concept	Description
	axioms: node(table, optimal_number_of_trees_t)
drainage_system_factor	Concept drainage_system_factor; sub_type_of: farm_factors; axioms: node(table, drainage_system_factor_t)
Organic_manure_factor	Concept organic_manure_factor; sub_type_of: plantation_factors; axioms: node(table , Organic_manure_factor_t)
hypotheical_or_concentration	Concept hypotheical_or_concentration; sub_type_of: organic_manure_factor; axioms: node(relation , hypotheical_or_concentration_r)
actual_or_concentration	Concept actual_or_concentration; Sub_type_of: hypotheical_or_concentration; axioms: node(pre_condition_function ,actual_or_concentration_pcf)
evaluation_absent	Concept evaluation_absent; sub_type_of: actual_or_concentration; axioms: node(function , evaluation_absent_f)
evaluation_quantity	Concept evaluation_quantity; sub_type_of: actual_or_concentration; axioms: node(function , evaluation_quantity_f)
Optimal_quantity_of_organic_manure	Concept Optimal_quantity_of_organic_manure; sub_type_of: evaluation_quantity; axioms: node(function , Optimal_quantity_of_organic_manure_t)

b) Relation

Relation	Description
actual_or_concentration_pcf	relation actual_or_concentration_pcf; properties: name, visited; argument-1: organic_manure; argument-2: evaluation_absent, evaluation_quantity; axioms : ***R1: If*** unknown (name of organic_manure) ***Then*** Visited of evaluation_absent = yes ***R2: If*** known (name of organic_manure) ***Then*** Visited of evaluation_quantity = yes
hypotheical_or_concentration_r	relation hypotheical_or_concentration_r; properties: value; argument-1: actual_or_concentration;

Relation	Description
	argument-2: hypotheical_or_concentration; axioms : ***R1: If*** (value of actual_or_concentration >200 , =<1000) ***Then*** Value of hypotheical_or_concentration = 1000 ***R2: If*** (value of actual_or_concentration >175 , =<200) ***Then*** Value of hypotheical_or_concentration = 200 ***R3: If*** (value of actual_or_concentration >150 , =<175 ***Then*** value of hypotheical_or_concentration = 175 ***R4: If*** (value of actual_or_concentration >125 , =<150 ***Then*** value of hypotheical_or_concentration = 150 ***R5: If*** (value of actual_or_concentration >100, =<125 ***Then*** value of hypotheical_or_concentration = 125 ***R6: If*** (value of actual_or_concentration >75 , =<100 ***Then*** value of hypotheical_or_concentration = 100 ***R7: If*** (value of actual_or_concentration >50 , =<75 ***Then*** value of hypotheical_or_concentration = 75 ***R8: If*** (value of actual_or_concentration >25 , =<50 ***Then*** value of hypotheical_or_concentration = 50 ***R9: If*** (value of actual_or_concentration >=0 , =<25 ***Then*** value of hypotheical_or_concentration = 25

Three) Table

drainage_system_factor **table** drainage_system_factor;
Input([farm.drainage_system])
Output([ds_factor.value]).

	Input	**output**	
Concept	Property	Concept	Property
farm	drainage_system	ds_factor	value
	good		1
	medium		0.9
	bad		0.8

organic_manure_factor_t **table** organic_manure_factor_t;
I**nput**([hypotheical_or_concentration.value,soil.type])
Output([organic_manure_factor.value]).

Input				Output	
concept hypotheical_or_concentration	**property value**	**concept soil**	**property type**	**concept Organic_manure_factor**	**property value**
	1000		Fine		1.05
	1000		Medium		1.075
	1000		Coarse		1.075
	200		Fine		1.05
	200		Medium		1.075
	200		Coarse		1.1
	175		Fine		1.05
	175		Medium		1.075
	175		Coarse		1.075
	150		Fine		1.05
	150		Medium		1.05
	150		Coarse		1.05
	125		Fine		1.025
	125		Medium		1.025
	125		Coarse		1.025
	100		Fine		0.975
	100		Medium		0.975
	100		Coarse		0.975
	75		Fine		0.95
	75		Medium		0.95
	75		Coarse		0.95
	50		Fine		0.95
	50		Medium		0.925
	50		Coarse		0.925
	25		Fine		0.95
	25		Medium		0.925
	25		Coarse		0.875

optimal_quantity_of_organic_manure_t **table** optimal_quantity_of_organic_manure_t;
I**nput**([organic_manure.name,farm.type,soil.type])
Output([optimal_quantity_of_organic_manure.value]).

input						output	
concept	**property**	**concept**	**property**	**concept**	**property**	**concept**	**property**
Organic_manure	**name**	**farm**	**type**	**soil**	**type**	optimal_quantity_of_organic_manure	**value**

input						output	
concept	**property**	**concept**	**property**	**concept**	**property**	**concept**	**property**
Organic_manure	**name**	**farm**	**type**	**soil**	**type**	optimal_quantity_of_organic_manure	**value**
	cattle		Open_field		fine		0.00275
	cattle		Open_field		medium		0.00372
	cattle		Open_field		coarse		0.00461
	hog		Open_field		fine		0.00275
	hog		Open_field		medium		0.00372
	hog		Open_field		coarse		0.00461
	chicken		Open_field		fine		0.001885
	chicken		Open_field		medium		0.00233
	chicken		Open_field		coarse		0.00275
	chicken bayade		Open_field		fine		0.001885
	chicken bayade		Open_field		medium		0.00233
	chicken bayade		Open_field		coarse		0.00275
	chicken magazer		Open_field		fine		0.001885
	chicken magazer		Open_field		medium		0.00233
	chicken magazer		Open_field		coarse		0.00275
	sheep		Open_field		fine		0.001885
	sheep		Open_field		medium		0.00233
	sheep		Open_field		coarse		0.00275
	hourse		Open_field		fine		0.00372
	hourse		Open_field		medium		0.00461
	hourse		Open_field		coarse		0.0055
	pigeon		Open_field		fine		0.001885
	pigeon		Open_field		medium		0.00275
	pigeon		Open_field		coarse		0.00372

expected_fruit_yield _t **table** expected_fruit_yield _t;
I**nput**([citrus.variety])
Output([expected_fruit_yield.value]).

input		output	
Concept	**Property**	**Concept**	**Property**
citrus	variety	optimal_no_of_plants_depend_on_variety	value
	balady		20
	limon		10
	naval		١5
	valancia		15

optimal_number_of_trees _t **table** optimal_number_of_trees _t;
I**nput**([citrus.variety])
Output([optimal_number_of_trees.value]).

input		output	
Concept	**Property**	**Concept**	**Property**
Citrus	variety	optimal_no_of_plants_depend_on_variety	value
	balady		١٦٨
	limon		٢٦٢
	naval		١٦٨
	valancia		١٦٨

d) Function

Function	Description
plantation_ factors_f	plantation_ factors.value= farm_factors.value* organic_manure_factor.value
normal_dimension_f	normal_dimension.value = 1
farm_factors_f	farm_factors.value= efy_factor.value*current_plant_factor.value * drainage_system_factor.value
efy_factor_f	efy_factor.value=citrus.afy/ expected_fruit_yield.value
current_plant_factor_f	current_plant_factor.value=(current_of_plant.no_of_trees/ farm.area)/ (optimal_number_of_trees.value/4200)
evaluation_absent_f	evaluation_absent.value=0
evaluation_quantity_f	(organic_manure.quantity/farm.area)/ optimal_quantity_of_organic_manure.value *100

6.1.1.5 Computional Ontology

4.1.1.5.1 Et0 Ontology

One) Et0 concept

Concept	Description
et0	Concept et0; Properties: Visited: NOMINAL, VALUE-LIST(yes;no), SOURCE(Derived) SINGLE¡ value: real, NUMBER-RANGE(0;1000), SOURCE(DERIVED) SINGLE. axioms: node(pre_condition_function , et0_pcf)
et0_pennman	Concept et0_pennman; sub_type_of: et0; axioms: node(pre_condition_function, et0_pennman_pcf)
l_rh	Concept l_rh;

Concept	Description
	sub_type_of: et0_pennman; axioms: node(function, l_rh_f)
m_rh	Concept m_rh; sub_type_of: et0_pennman; axioms: node(function, m_rh_f)
h_rh	Concept h_rh ; sub_type_of: et0_pennman; axioms: node(function, h_rh_f)

Two) Et0 Relation

Relation	Description
et0_pcf	relation et0_pcf; properties: type, visited; argument-1: farm; argument-2: et0_pennman; axioms : ***R2: If*** type of farm= open_field ***Then*** visited of et0_ pennman = yes
et0_pen_pcf	relation et0_pen _pcf; properties: max_rh(task_parameters.current_month),visited; argument-1: climate; argument-2: l_rh, m_rh, h_rh; axioms : ***R1: If*** max_rh(task_parameters. current _month) of climate <= 30 ***Then*** visited of l_rh = yes ***R2: If*** max_rh(task_parameters. current _month) of climate (> 30 <=60) ***Then*** visited of m_rh = yes ***R3: If*** max_rh(task_parameters. current _month) of climate > 60 ***Then*** visited of h_rh = yes

c) Et0 function

Function	Description
l_rh_f	l_rh_c.value= (ra.value*(0.25+0.5*(climate.msh(task_parameters.current_month)/ece.value)))*(((

Function	Description
	climate.dws(task_parameters.current_month)/3.6)*(0.003872-0.000364*(climate.dws(task_parameters.current_month)/ climate.nws(task_parameters.current_month))))+(0.000133*(climate.dws(task_parameters.current_month)/ climate.nws(task_parameters.current_month))+0.007167))+((climate.dws(task_parameters.current_month)/3.6)*(0.011283*(climate.dws(task_parameters.current_month) / climate.nws)-0.08775))+(1.0425-0.0185*(climate.dws(task_parameters.current_month) / climate.nws(task_parameters.current_month))) *((0.440904+0.0000319*farm.altitude+ climate.avg_tc(task_parameters.current_month)*(0.011221-0.00000048* farm.altitude))*(0.75* ra_l.value *(0.25+0.5*(climate.ash(task_parameters.current_month) /(climate.msh(task_parameters.current_month)))-((10.8158+ climate.avg_tc(task_parameters.current_month)*0.1965)*(0.34-0.044*(EXP(1.872543+0.06237* climate.avg_tc(task_parameters.current_month)) * climate.avg_rh(task_parameters.current_month)/100)^0.5)* (0.1+0.9* (climate.ash(task_parameters.current_month)/(climate. msh(task_parameters.current_month))))))+(1-(0.440904+0.0000319*farm.altitude + climate.avg_tc(task_parameters.current_month)*(0.011221-0.00000048* farm.altitude)))*(0.27+0.0648* climate.mws(task_parameters.current_month))*(EXP(1.872543+0.06237* climate.avg_tc(task_parameters.current_month))*(1- climate.avg_rh(task_parameters.current_month)/100)))
m_rh_f	m_rh_c.value= ra.value *(((climate.dws(task_parameters.current_month)/3.6)*(0.003428-0.000022*(climate.dws(task_parameters.current_month)/ climate.nws(task_parameters.current_month))))+(0.00117*(climate.dws(task_parameters.current_month)/ climate.nws(task_parameters.current_month))+0.001167))+((climate.dws (task_parameters.current_month)/3.6)*(0.010033*(climate.dws(task_parameters.current_month)/ climate.nws(task_parameters.current_month))-0.072583))+(1.125-0.0255*(climate. dws(task_parameters.current_month)/ climate.nws(task_parameters.current_month))) *((0.440904+0.0000319*farm.altitude+ climate.avg_tc(task_parameters.current_month)*(0.011221-0.00000048* farm.altitude))*(0.75* ra.value *(0.25+0.5*(climate.ash(task_parameters.current_month) /(climate.msh(task_parameters.current_month)))-((10.8158+ climate.avg_tc(task_parameters.current_month)*0.1965)*(0.34-

Function	Description
	0.044*(EXP(1.872543+0.06237* climate.avg_tc(task_parameters.current_month)) * climate.avg_rh(task_parameters.current_month)/100)^0.5)* (0.1+0.9* (climate.ash(task_parameters.current_month)/(climate. msh(task_parameters.current_month))))))+(1-(0.440904+0.0000319*farm.altitude + climate.avg_tc(task_parameters.current_month)*(0.011221-0.00000048* farm.altitude)))*(0.27+0.0648* climate.mws(task_parameters.current_month))*(EXP(1.872543+0.06237* climate.avg_tc(task_parameters.current_month))*(1- climate.avg_rh(task_parameters.current_month)/100)))
h_rh_f	h_rh_c.value= (ra.value*(0.25+0.5*(climate.ash(task_parameters.current_month) / ece.value)))*(((climate.dws(task_parameters.current_month) /3.6)*(0.000139*(climate.dws(task_parameters.current_month)/ climate.nws(task_parameters.current_month))+0.003122))+(0.00185*(climate.dws(task_parameters.current_month)/ climate.nws(task_parameters.current_month))-0.000083))+((climate.dws(task_parameters.current_month) /3.6)*(0.009333*(climate.dws(task_parameters.current_month) / climate.nws(task_parameters.current_month))-0.0645))+(1.16625-0.026*(climate.dws(task_parameters.current_month)/ climate.nws(task_parameters.current_month))))))*((0.440904+0.0000319*farm.altitude+ climate.avg_tc(task_parameters.current_month)*(0.011221-0.00000048* farm.altitude))*(0.75* ra.value *(0.25+0.5*(climate.ash(task_parameters.current_month) /(climate.msh(task_parameters.current_month))))-((10.8158+ climate.avg_tc(task_parameters.current_month)*0.1965)*(0.34- 0.044*(EXP(1.872543+0.06237* climate.avg_tc(task_parameters.current_month)) * climate.avg_rh(task_parameters.current_month)/100)^0.5)* (0.1+0.9* (climate.ash(task_parameters.current_month)/(climate. msh(task_parameters.current_month))))))+(1-(0.440904+0.0000319*farm.altitude + climate.avg_tc(task_parameters.current_month)*(0.011221-0.00000048* farm.altitude)))*(0.27+0.0648* climate.mws(task_parameters.current_month))*(EXP(1.872543+0.06237* climate.avg_tc(task_parameters.current_month))*(1- climate.avg_rh(task_parameters.current_month)/100)))

4.1.1.6.2 Eta Ontology

a)Eta concept

Concept	Description
eta	Concept eta; Properties: Visited: NOMINAL, VALUE-LIST(yes;no), SOURCE(Derived) SINGLE; value: real, NUMBER-RANGE(0;1000), SOURCE(DERIVED) SINGLE. axioms: node(function , eta_f)
gc	Concept gc ; sub_type_of: eta; axioms: node(pre_condition_function, gc_pcf)
gc_balady	Concept gc_balady; sub_type_of: gc; axioms: node(function, gc_balady _f)
gc_ limon	Concept gc_ limon; sub_type_of: gc; axioms: node(function, gc_ limon _f)
gc_ naval	Concept gc_ naval; sub_type_of: gc; axioms: node(function, gc_ naval _f)
gc_ valancia	Concept gc_ valancia; sub_type_of: gc; axioms: node(function, gc_ valancia _f)
kc	Concept kc; sub_type_of: eta; axioms: node(pre_condition_function, kc_pcf)
kc_ init_st	Concept kc_ init_st; sub_type_of: kc; axioms: node(function, kc_ init_st_f)
kc_ ve_st	Concept kc_ ve_st; sub_type_of: kc; axioms: node(function, kc_ ve_st_f)
kc_ fl_st	Concept kc_ fl_st;

Concept	Description
	sub_type_of: kc; axioms: node(function, kc_ fl_st_f)
kc_ fr_st	Concept kc_ fr_st; sub_type_of: kc; axioms: node(function, kc_ fr_st_f)
kc_ wint_per	Concept kc_ wint_per; sub_type_of: kc; axioms: node(function, kc_ wint_per _f)
depression_factor	Concept depression_factor; sub_type_of: eta; axioms: node(relation, depression_factor_r)
unit_factor	Concept unit_factor; sub_type_of: eta; axioms: node(relation, unit_factor_r)

Two) Eta Relation

Relation	Description
gc_pcf	relation gc_pcf; properties: variety, visited; argument-1: citrus; argument-2: gc_balady, gc_limon, gc_naval, gc_valancia; axioms : ***R1: If*** .variety of citrus = balady ***Then*** visited of gc_balady = yes ***R2: If*** variety of citrus = limon ***Then*** visited of gc_limon = yes ***R3: If*** variety of citrus =naval ***Then*** visited of gc_naval = yes ***R4: If*** variety of citrus = valancia ***Then*** visited of gc_valancia= yes
kc_pcf	relation kc_pcf; properties: growth_stage, visited; argument-1: citrus; argument-2: kc_init_st, kc_ve_st, kc_fl_st, kc_fr_st, kc_wint_per; axioms : ***R1: If*** growth_stage of citrus = init_stage ***Then*** visited of kc_init_st = yes

Relation	Description
	R2: If growth_stage of citrus = ve_stage ***Then*** visited of kc_ve_st = yes ***R3: If*** growth_stage of citrus = fl_stage ***Then*** visited of kc_fl_st = yes ***R4: If*** growth_stage of citrus = fr_stage ***Then*** visited of kc_fr_st = yes ***R5: If*** growth_stage of citrus = wint_per ***Then*** visited of kc_wint_per = yes
depression_factor_r	relation depression_factor_r; properties: type, value; argument-1: farm; argument-2: depression_factor; axioms : ***R1: If*** type of farm = open_field ***Then*** value of depression_factor = 1
unit_factor_r	relation unit_factor_r; properties: type, value; argument-1: farm; argument-2: unit_factor; axioms : ***R1: If*** type of farm = open_field ***Then*** value of unit_factor = 4200

c) Eta function

Function	Description
eta_f	eta.value= depression_factor.value*(unit_factor.value/1000) kc.value*et0.value*(gc.value/100)
gc_balady _f	gc_balady .value =(0.0061* current_planting age^3-0.4514* current_planting age^2+10.753* current_planting age+5.5161);
gc_limon_f	gc_limon.value =(0.0061* current_planting .age^3-0.4514* current_planting .age^2+10.753* current_planting .age+10.516)))
gc_naval_f	gc_naval.value =(0.0061* current_planting .age^3-0.4514* current_planting .age^2+10.753* current_planting .age+10.516)))
gc_valancia_f	gc_valancia.value =(0.0061* current_planting .age^3-0.4514*age^2+10.753* current_planting .age+10.516)))

Kc_init_st_f	kc_init_st.value =(0.00003* current_planting.age^3-0.0019* current_planting.age ^2+0.0448* current_planting.age +0.4605);
kc_ve_st_f	kc_ve_st.value =(0.00003* current_planting.age^3-0.0019* current_planting.age ^2+0.0448* current_planting.age +0.5605)
kc_ fl_st_f	kc_ fl_st.value = (0.00003* current_planting.age^3-0.0019* current_planting.age ^2+0.0448* current_planting.age +0.5605);
kc_ fr_st_f	kc_ fr_st.value =(0.00003* current_planting.age^3-0.0019* current_planting.age ^2+0.0448* current_planting.age +0.6605);
Kc_wint_per_f	kc_wint_per.value=(0.00003* current_planting.age^3-0.0019* current_planting.age ^2+0.0448* current_planting.age +0.3605)

4.1.1.6.3 Pawc Ontology

a)Pawc concept

Concept	Description
pawc	Concept pawc; Properties: Visited: NOMINAL, VALUE-LIST(yes;no), SOURCE(Derived) SINGLE; value: real, NUMBER-RANGE(0;1000), SOURCE(DERIVED) SINGLE. axioms: node(function , pawc_f)
ad	Concept ad; sub_type_of: pawc; axioms: node(relation, ad_r)
adaptive_rd	Concept adaptive_rd; sub_type_of: pawc; axioms: node(relation, adaptive_rd _r)
rd	Concept rd; sub_type_of: irr_system_factor axioms: node(pre_condition_function , rd_pcf)
rd_f_d	Concept rd_f_d sub_type_of: rd; axioms: node(pre_condition_function , rd_f_d_f)
rd_m_d	Concept rd_m_d

Concept	Description
	sub_type_of: rd; axioms: node(pre_condition_function , rd_m_d_f)
rd_c_d	Concept rd_c_d sub_type_of: rd; axioms: node(pre_condition_function , rd_c_d_f)
rd_f_f	Concept rd_f_f sub_type_of: rd; axioms: node(pre_condition_function , rd_f_f_f)
rd_m_f	Concept rd_m_f sub_type_of: rd; axioms: node(pre_condition_function , rd_m_f_f)
rd_c_f	Concept rd_c_f sub_type_of: rd; axioms: node(pre_condition_function , rd_c_f_f)
rd_f_s	Concept rd_f_s sub_type_of: rd; axioms: node(pre_condition_function , rd_f_s_f)
rd_m_s	Concept rd_m_s sub_type_of: rd; axioms: node(pre_condition_function , rd_m_s_f)
rd_c_s	Concept rd_c_s sub_type_of: rd; axioms: node(pre_condition_function , rd_c_s_f)
unit_factor	Concept unit_factor; sub_type_of: pawc; axioms: node(relation , unit_factor_r)
sp_factor	Concept sp_factor; sub_type_of: pawc; axioms: node(pre_condition_function , sp_factor_pcf)
sp_l	Concept sp_l; sub_type_of: sp_factor; axioms: node(function , sp_l_f)
sp_h	Concept sp_h; sub_type_of: sp_factor;

Concept	Description
	axioms: node(function , sp_h_f)

b) Pawc Relation

Relation	Description
	relation ad_r; properties: caco3,age; argument-1: soil,current_planting; argument-2: ad; axioms : ***R1: If*** age of current_planting < 4 Caco3 of soil < 2 ***Then*** value of ad = 0.25 ***R2: If*** age of current_planting < 4 Caco3 of soil >=2 Caco3 of soil < 5 ***Then*** value of ad = 0.15 ***R3: If*** age of current_planting < 4 Caco3 of soil >= 5 Caco3 of soil < 10 ***Then*** value of ad = 0.1 ***R4: If*** age of current_planting < 4 Caco3 of soil >= 10 ***Then*** value of ad = 0.5 ***R5: If*** age of current_planting >= 4 Caco3 of soil < 2 ***Then*** value of ad = 0.25 ***R6: If*** age of current_planting .>= 4 Caco3 of soil >= 2 Caco3 of soil < 5 ***Then*** value of ad = 0.25 ***R7: If*** age of current_planting >=4 Caco3 of soil >= 5 Caco3 of soil < 10 ***Then*** value of ad = 0.2 ***R8: If*** age of current_planting >= 4 Caco3 of soil >= 10 ***Then*** value of ad = 0.15

Relation	Description
adaptive_rd _r	relation adaptive_rd _r; properties: method , value; argument-1: irrigation; argument-2: adaptive_rd; axioms : ***R1: If*** method of irrigation = flooding Value of rd >= 100 ***Then*** value of adaptive_rd = 100/100 ***R2: If*** method of irrigation = Sprinkler Value of rd >= 80 ***Then*** value of adaptive_rd = 80/100 ***R3: If*** method of irrigation = drip Value of rd >= 60 ***Then*** value of adaptive_rd = 60/100 ***R4: If*** method of irrigation = flooding Value of rd < 100 ***Then*** value of adaptive_rd = rd.value/100 ***R5: If*** method of irrigation = Sprinkler Value of rd < 80 ***Then*** value of adaptive_rd = rd.value/100 ***R6: If*** method of irrigation = drip Value of rd < 60 ***Then*** value of adaptive_rd = rd.value/100
rd_pcf	relation rd_pcf; properties: age, init_stage, init_ve_stage, init_ve_fl_stage agriculture_method, visited; argument-1: task_parameters, plant, current_planting; argument-2: rd_init_st, rd_ve_st, rd_fl_st, rd_fr_st; axioms : ***R1: If*** type of soil = fine method of irrigation = drip ***Then*** visited of rd_f_d = yes ***R2: If*** type of soil = medium method of irrigation = drip ***Then*** visited of rd_m_d = yes ***R3: If*** type of soil = coarse method of irrigation = drip ***Then*** visited of rd_f_d= yes

Relation	Description
	R4: If type of soil = fine method of irrigation = flooding ***Then*** visited of rd_f_f = yes ***R5: If*** type of soil = medium method of irrigation = flooding ***Then*** visited of rd_m_f = yes ***R6: If*** type of soil = coarse method of irrigation = flooding ***Then*** visited of rd_f_f= yes ***R7: If*** type of soil = fine method of irrigation = sprinkler ***Then*** visited of rd_f_s = yes ***R8: If*** type of soil = medium method of irrigation = sprinkler ***Then*** visited of rd_m_s = yes ***R9: If*** type of soil = coarse method of irrigation = sprinkler ***Then*** visited of rd_f_s= yes
unit_factor_r	relation unit_factor_r; properties: sp, visited; argument-1: soil; argument-2: sp_l,sp_g; axioms : ***R1: If*** type of farm = open_field ***Then*** value of unit_factor = 4200
sp_factor_pcf	relation sp_factor_pcf; properties: sp, visited; argument-1: soil; argument-2: sp_l,sp_g; axioms : ***R1: If*** sp of soil < 37 Sp < 37 ***Then*** visited of sp_l = yes ***R2: If*** sp of soil >= 37 ***Then*** visited of sp_h = yes

c) Pawc table

d) Pawc function

Function	Description
pawc_f	Pawc.value= ad.value* (adaptive_rd.value/100)*soil.spd*sp_factor.value* unit_factor.value
Rd_f_d_f	Rd_f_d_f .value= (0.003*current_planting.age^3-0.2257* current_planting.age ^2+5.3763* current_planting.age +35.258)
Rd_m_d_f	Rd_m_d_f .value= (0.0041* current_planting.age ^3-0.3009* current_planting.age ^2+7.1685* current_planting.age +33.677)
Rd_c_d_f	Rd_c_d_f .value= (0.0051* current_planting.age ^3-0.3761* current_planting.age ^2+8.9606* current_planting.age +32.097)
Rd_f_f_f	Rd_f_f_f .value= (0.003* current_planting.age^3-0.2257* current_planting.age ^2+5.3763* current_planting.age +35.258)
Rd_m_f_f	Rd_m_f_f .value= (0.0041* current_planting.age ^3-0.3009* current_planting.age ^2+7.1685* current_planting.age +33.677)
Rd_c_f_f	Rd_c_f_f .value= (0.0051* current_planting.age ^3-0.3761* current_planting.age ^2+8.9606* current_planting.age +32.097)
Rd_f_s_f	Rd_f_s_f .value= (0.003* current_planting.age ^3-0.2257* current_planting.age ^2+5.3763* current_planting.age+35.258)
Rd_m_s_f	Rd_m_s_f .value= (0.0041* current_planting.age^3-0.3009* current_planting.age^2+7.1685* current_planting.age+33.677)
Rd_c_s_f	Rd_c_s_f .value= (0.0051* current_planting.age^3-0.3761* current_planting.age^2+8.9606* current_planting.age+32.097)

4.1.1.6.4 Interval Ontology

a)Interval concept

Concept	Description
interval	Concept interval; Properties: Visited: NOMINAL, VALUE-LIST(yes;no), SOURCE(Derived) SINGLE; value: real, NUMBER-RANGE(0;1000), SOURCE(DERIVED) SINGLE. default-value-def : value = 1; axioms: node(pre_condition_function , interval_pcf)
weekly_basis_interval	Concept weekly_basis_interval; sub_type_of: interval; axioms: node(pre_condition_function, weekly_basis_interval_pcf)
interval_weekly	Concept interval_weekly; sub_type_of: weekly_basis_interval; axioms: node(function, interval_weekly_f)
user_suggested_interval_ weekly	Concept user_suggested_interval_weekly; sub_type_of: weekly_basis_interval; axioms:

Concept	Description
	node(function, user_suggested_interval_weekly_f)

b) Interval Relation

Relation	Description
interval_pcf	relation interval_pcf; properties: schedule_type,visited; argument-1: irrigation; argument-2: control_water_weekly_basis,plant_age; axioms : ***R1: If*** schedule_type of irrigation = monthly ***Then*** visited of control_water_weekly_basis = yes
weekly_basis_interval_pcf	relation weekly_basis_interval_pcf; properties: controled_water,visited; argument-1: irrigation; argument-2: interval_weekly,user_suggested_interval_weekly axioms : ***R1: If*** controled_water of irrigation = yes ***Then*** visited of interval_weekly = yes ***R2: If*** controled_water of irrigation = no ***Then*** visited of user_suggested_interval_weekly = yes

c) interval function

Function	Description
interval_weekly_f	interval_weekly.value= pawc.value/eta.value
user_suggested_interval_weekly_f	user_suggested_interval_weekly.value= irrigation.user_suggested_interval

4.1.1.6.5 Water_requirement Ontology
a) water_requirement concept

Concept	Description
water_requirement	Concept water_requirement; Properties: Visited: NOMINAL, VALUE-LIST(yes;no), SOURCE(Derived) SINGLE*j* value: real, NUMBER-RANGE(0;1000),

Concept	Description
	SOURCE(DERIVED) SINGLE. axioms: node(function , water_requirement_f)
adaptive_lr	Concept adaptive_lr; sub_type_of: water_requirement; axioms: node(post_condition_function, adaptive_lr_pocf)
lr	Concept lr; sub_type_of: adaptive_lr; axioms: node(function, lr_f)
ece	Concept ece; sub_type_of: lr ; axioms: node(table, ece_t)
	Concept water_used; sub_type_of: water_requirement; axioms: node(pre_condition_function, water_used_pcf)
water_used_monthly	Concept water_used_monthly; sub_type_of: water_used; axioms: node(function, water_used_monthly_f)

b) Water Requirement Relation

Relation	Description
adaptive_lr_pocf	Relation adaptive_lr_pocf; properties: value; argument-1: lr; argument-2: adaptive_lr; axioms : ***R1: If*** value of lr > 0.25 ***Then*** value of adaptive_lr = 0.25 ***R2: If*** value of lr <= 0.25 ***Then*** value of adaptive_lr = value of lr
water_used_pcf	relation water_used_pcf; properties: schedule_type,visited; argument-1: irrigation; argument-2: water_used_weekly, water_used_daily; axioms : ***R2: If*** schedule_type of irrigation = monthly ***Then*** visited of water_used_weekly = yes

c) water requirement ***table***

ece_t **table** ece_t;
Input([citrus.variety])
Output([ece.value]).

input		output	
citrus	**variety**	**ece**	value
	balady		2.2
	Limon		2.5
	naval		1.7
	valancia		1.7

d) water_requirement function

Function	Description
water_requirement_f	water_requirement= ((water_used_monthly.value / irrigation.irrigation_efficiency) *(1+lr.value) * plantation_factor.value
lr_f	lr.value=water.eciw/ece.value
water_used_weekly_f	water_used_monthly.value=interval.value*eta.value

4.1.1.6.5 Irrigation_units Ontology

a) irrigation_units Concept

Concept	Description
irrigation_units	Concept irrigation_units; Properties: Visited: NOMINAL, VALUE-LIST(yes;no), SOURCE(Derived) SINGLE; Value: real, NUMBER-RANGE(0;1000), SOURCE(DERIVED) SINGLE. Value_n: nominal, VALUE-LIST(light_irrigation, medium_irrigation, heavy_irrigation), SOURCE(DERIVED[relation, irrigation_type_r]) SINGLE. axioms: node(pre_condition_function, property_of_irrigation_system _pcf)
flooding_irrigation	Concept flooding_irrigation; sub_type_of: irrigation_units; axioms: node(aggregation, _)
irrigation_type	Concept irrigation_type;

Concept	Description
	sub_type_of: flooding_irrigation; axioms: node(relation, irrigation_type_r)

b) Irrigation_units Relation

Relation	Description
irrigation_units_pcf	relation irrigation_units_pcf; properties: method,visited; argument-1: irrigation; argument-2: flooding_irrigation,drip_irrigation axioms : ***R1: If*** (method of irrigation = flooding ***Then*** visited of flooding_irrigation = yes
irrigation_type_r	relation irrigation_type_r; properties: water_requirement, value,value_n; argument-1: irrigation_schedule; argument-2: irrigation_type; axioms : ***R1: If*** water_requirement of irrigation_schedule (< 200) ***Then*** value_n of irrigation_type = light_irrigation ***R2: If*** water_requirement of irrigation_schedule (>= 200, <=300) ***Then*** value_n of irrigation_type = medium_irrigation ***R3: If*** water_requirement of irrigation_schedule (> 300) ***Then*** value_n of irrigation_type = heavy_irrigation

Domain Model

A domain model is defined as a coherent collection of expressions about a domain that represents a particular viewpoint defined in an ontology. The domain model may therefore embody certain assumptions that are specific for the ontology that is used.

4.2.1Expansion Model

DOMAIN-MODEL expansion model
USES: plantation factors ontology
TYPE: Dependency graph
PARTS:
plantation_factors: dependency graph.

4.2.2 Computional Model

DOMAIN- MODEL computional model
USES: computional ontology

TYPE: dependency graph
PARTS: et0 : dependency graph ,
eta : dependency graph ,
pawc : dependency graph,
interval : dependency graph ,
water_requirement : dependency graph,

6. Control Knowledge

As shown in figure 2, the task control knowledge consists of three basic parts task knowledge, inference knowledge and library of primitive problem solver. The task knowledge composed of one sub-tasks namely: "*propose"*. All primitive sub-task mapped into a set of inference steps. The following sub-section discuses Inference Knowledge , task knowledge, library of intelligent problem solvers.

5.1 Inference Knowledge

The design of inference knowledge consists of two main parts namely: inference structure and inference specification. The following paragraphs explain them in much more details.

5.1.1 Inference Structure

As shown in the following figure the inference structure includes three inference steps. The objective of the *expand* inference is to use known data to derive new ones using a set of relations that forms the *expansion model*. The goal of propose irrigation schedule is to get the results of the expand inference step and use the computational model to generate a irrigation schedule.

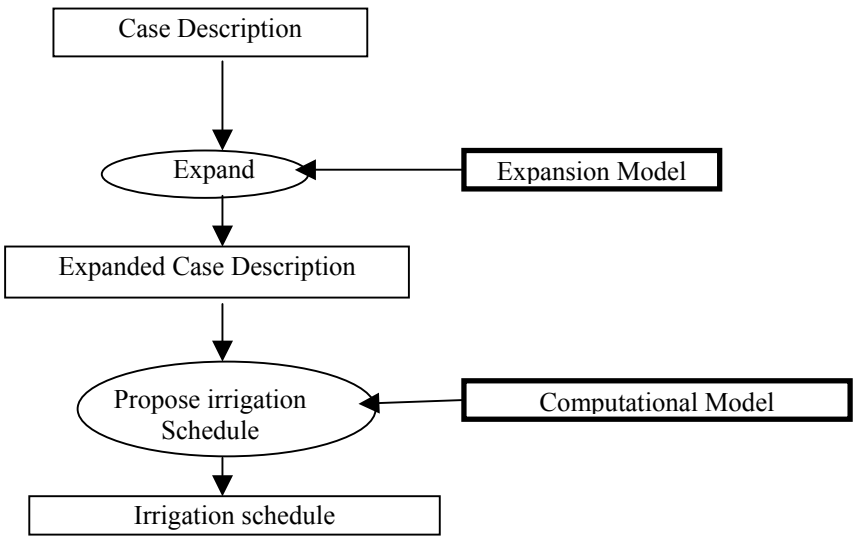


Figure 10: Inference Structure for the Irrigation Schedule

5.1.2 Inference specification

Names of inferences represent the role that these inferences play in solving the problem. Inference names are thus goal-oriented. For each role, a mapping is specified to the domain knowledge. For instance, static roles indicate which domain model should be accessed. Dynamic roles, on the other hand, are supposed to be part of the overall working memory of the problem solver and are thus not directly linked to specific domain model. Two inference steps from the irrigation application are given in figure 11.

Inference	expand
operation-type	Compute
input-roles	case description
output-roles	expanded case description
static-roles	expansion model
methods	Mechanism-selector(+expansion model , +part identifier, - mechanism name) execute(+mechanism name, +part identifier)
operation-type	Compute
input-roles	expanded case description
output-roles	Initial irrigation schedule
static-roles	Computational model
methods	**get** *Part-list of* Computational *_model* **while***(not empty-list)* **Begin** *part from list* **Case** *part* *et0:* **Mechanism-selector**(+et0_ model , +part identifier, - mechanism name) **execute**(+mechanism name, +part identifier) *eta:* **Mechanism-selector**(+eta_model , +part identifier, - mechanism name) **execute**(+mechanism name, +part identifier) *pawc:* **Mechanism-selector**(+pawc_model , +part identifier, - mechanism name) **execute**(+mechanism name, +part identifier) *interval:* **Mechanism-selector**(+interval_model , +part identifier, - mechanism name) **execute**(+mechanism name, +part identifier) *water_requirement:* **Mechanism-selector**(+water_requirement_model , +part identifier, - mechanism name) **execute**(+mechanism name, +part identifier) **EndCase** **End***{while}*

Figure 11: Inference Steps for the Irrigation Schedule

5.2 Task knowledge

5.2.1 Main Task Specification

Figure 12 shows the task Structure for the irrigation schedule. The task structure consists of three parts input data handeling, task process and output data handeling. The input and output data handeling mapped to input and output transfer task respectively. The task process composed of one sub-tasks namely: "*propose*"and one procedure "verify".

Figure 13.1 shows the irrigation scheduale task The task consists of two parts: the *task definition* and *the task body*. The task definition describes the main goal of irrigation schedule as well as the input, and the output roles. The task body describes the control over sub-task. Figure 13.2 shows the sub-task. Figure 13.3 describes the input/ output transfer tasks. The transfer tasks are representing the interaction with the user. Finaly, the procedure of irrigation tasks represents in figure 13.4.

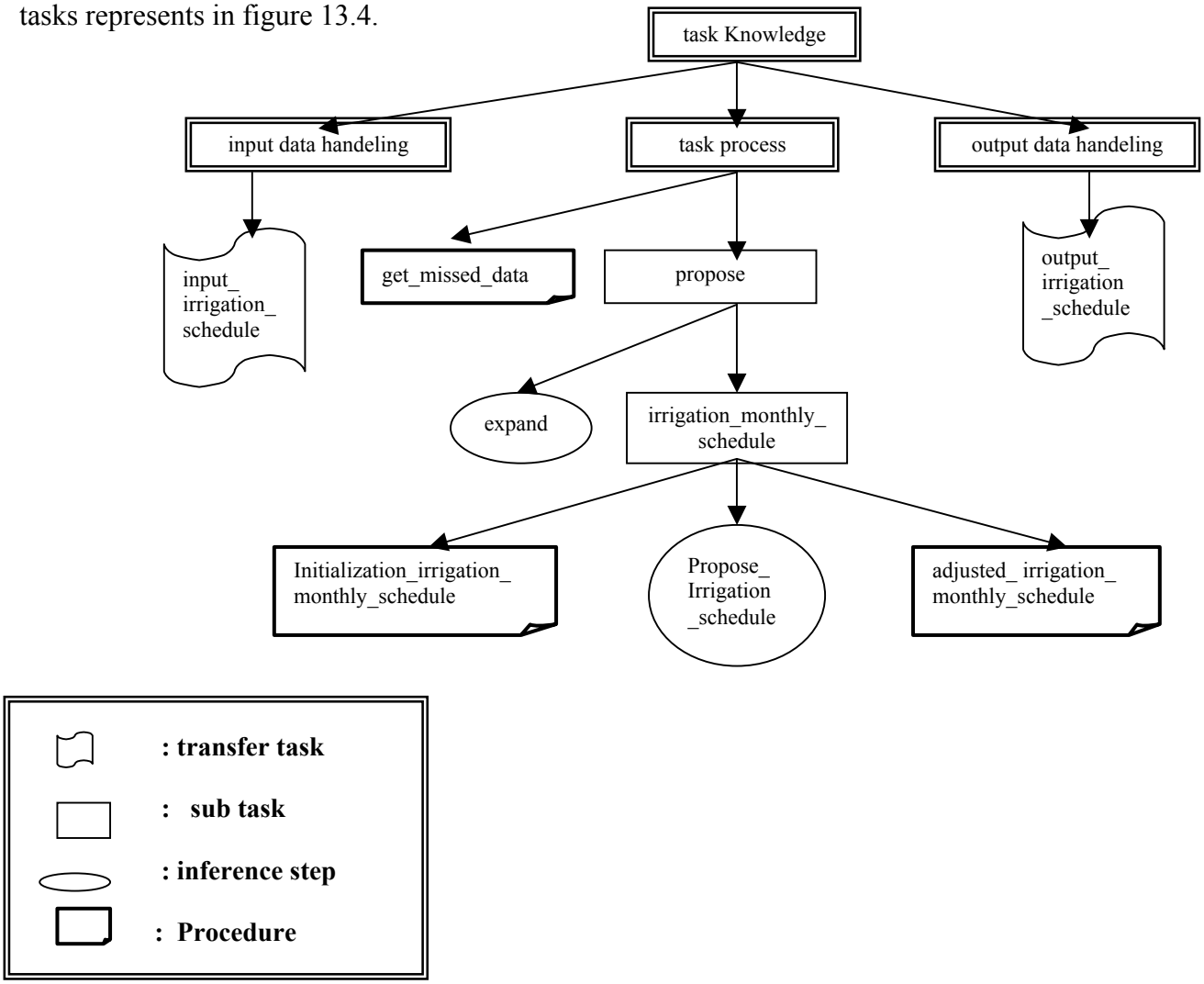


Figure 12: Task Structure for the Irrigation Schedule

Task	Description
irrigation_schedule	**task** irrigation schedule(+id) **task-definition** **goal:** the main goal of the irrigation is to determine the water requirement during cultivation. **input:** case-description **output:**irrigation schedule **task-body** **type:** composite **sub-tasks:** propose **transfer_tasks:** input_irrigation_schedule, output_irrigation_schedule. **procedure:** get_missed_data **control-structure:** ***Propose-schedule***(case description → irrigation schedule) = ***Iutput_irrigation_schedule*** ***Get_missed_data*** ***propose***(+case-description, - irrigation-schedule) ***output_irrigation_schedule***(irrigation_schedule)

Figure 13.1 : The Task Irrigation Schedule

Sub Task	Description
propose	**task** propose **task-definition** **goal:** Generating an irrigation schedule **input:** case description **output:** irrigation schedule **task-body** **type:** composite **additional-roles:** expanded-case-description **procedure:** irrigation_monthly_schedule. **control-structure:** ***expand***(+case description, + expansion model , -expanded-case-description), ***irrigation_monthly_schedule***.
irrigation_ monthly_ schedule	**task** irrigation_monthly_schedule **task-definition** **goal:** Generating an irrigation schedule **input:** expanded-case-description **output:** irrigation schedule **task-body** **type:** composite **procedure:** initialization_ irrigation_monthly_schedule adjusted_ irrigation_monthly_schedule **control-structure:** ***initialization_ irrigation_monthly_schedule*** **Repeat** ***get***(task_parameters.month(Month)), ***get***(citrus.variety(Variety)), ***look_up***(growth_stage_t(+Month, + Citrus, - Growth_stage) %execute table citrus.growth_stage = Growth_stage ***Propose-irrigation_schedule***(+expanded_case_description, +computional_model, -rrigation_schedule) **assert**(irrigation_schedule,[task_parameters.month, water_requirement.value, interval.value) ***adjusted_ irrigation_monthly_schedule*** **Until**(task_parameters.month>12

Figure 13.2 : The Sub Task Irrigation Schedule

Trasnsfer Task	Description
input_irrigation_ schedule	**Transfer_task** input_irrigation_schedule **control-structure:** ***input_irrigation_screen***

output_irrigation_ schedule	**Transfer_task** output_irrigation_schedule(+reccommendation, +irrigation_schedule)
	control-structure:
	get(irrigation_schedule_list of irrigation_schedule)
	If(irrigation.method=flooding) **then**
	While ***(not empty irrigation_schedule_list***)
	Begin
	Mechanism-selector(+irrigation_units_model , _, - mechanism name)
	Execute(+mechanism name, + irrigation_units_model)
	update(irrigation_schedule, [_,_ ,_, irrigation_type.value_n])
	End {while}
	Endif
	If(irrigation.method=flooding) **then**
	calling ***flooding_irrigation_screen***(irrigation-schedule)
	Else
	calling ***drip_sprinkler_irrigation_screen***(irrigation-schedule)
	EndIf

figure 13.3 : The TransferTask Irrigation Schedule

procedure	Description

get_missed_data	**Procedure** get_missed_data
	Begin
	get *soil_data_list of soil_data*
	get_value*(soil_data_list)*
	get *climate_data_list of climate_data*
	get_value(*climate_data_list)*
	get *irrigation_data_list of irrigation_data*
	task_parameters. current_month = 1
	Repeat
	get_value(*irrigation_data_list)*
	task_parameters. current_month = task_parameters. current_month + 1
	Until(task_parameters. current_month > 12)
	
	get_value(data_list):-
	While(not empty data_list[concept, property, type, name])
	Begin
	If(unknown(concept.property)) **Then**
	Case type
	table: look_up(name)
	relation: implication(name)
	function: equation(name)
	Endcase
	EndIf
	End*{while}*
	
	soil_data(soil, type, relation, soil_type).
	soil_data(soil, sp, table, sp_t).
	soil_data(soil, spd, table, sbd_t) .
	
	climate_data(climate, ra_par_a, table, ra_t).
	climate_data(climate, ra_par_b, table, ra_t).
	climate_data(climate, ra, function, ra_f).
	climate_data(climate, msh_par_a, table, msh_t).
	climate_data(climate, msh_par_b, table, msh_t).
	climate_data(climate, msh, function, msh_f).
	
	irrigation_data(irrigation, irrigation_efficiency, table, irrigation_efficiency_t).
	irrigation_data(irrigation, saa, table, saa_t).

initialization_ irrigation_ monthly_ schedule	**procedure** initialization_ irrigation_monthly_schedule **control-structure:** task_parameters.month = 1, type of farm = open_field, schedule_type of irrigation = monthly.
adjusted_ irrigation_ monthly_ schedule	**procedure** adjusted_ irrigation_monthly_schedule **control-structure:** task_parameters.month = task_parameters.month + 1,

Figure 13.4 : The ProcedureTask Irrigation Schedule

5.2.2 Transfer Task Specification

Transfer tasks are used to handle system transaction. Two types of transaction are designed input transaction in which the user can enter his/her data into the system where as output transaction are used to display the result obtain from using the system. The items representing the input transaction are vary depending on the user situation for instance, if the user run the system in a weekly bases mode the item including in the transaction should not include the average temperature, average relative humidity, actual sun shine hours, maximum sun shine hours, extra radiation, maximum relative humidity, mean wind speed, day wind speed and night wind speed .

Therefore screens are designed to group or items that are related to the current user context. These are reflected in the procedure below in this design. In this application, there are two types of screens: *input screen* (for collecting user data prior to the irrigation system execution) and *output screens* (which are displayed at the end of session). The input screen is shown in figure 14. There are two output screens for displaying the irrigation schedule concerning the drip and sprinkler irrigation in a monthly bases mode and flooding irrigation in a monthly bases mode. These screens are shown in figures 15, 16, and 17 respectively.

5.2.2.1 Screen Design

- **Input irrigation screen**

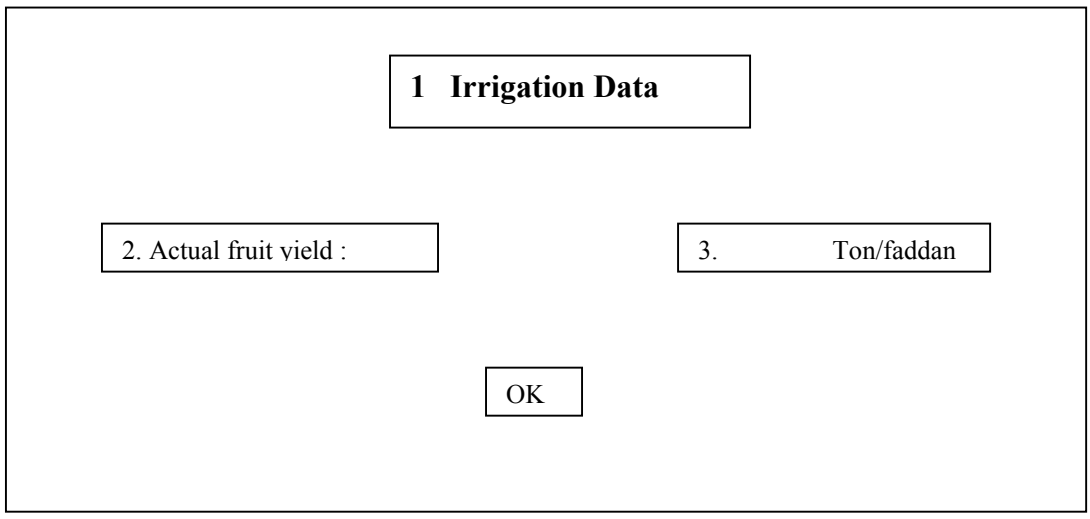


Figure 14: Irrigation Data Dialogue

- **Drip_sprinkler irrigatiom screen**

Irrigation Schedule

Month	water quantity M^3/faddan	interval
1	145	16
2	148	9
3	156	7
4	135	4
5	147	4
6	154	4
7	149	4
8	150	6
9	153	9
10	152	11
11	144	13
12	145	18

Figure 14: Drip Irrigation Screen

- **Flooding irrigatiom screen**

Irrigation Schedule

Month	water quantity M^3/faddan	interval	irrigation type
1	145	20	Light irr.
2	144	11	Light irr.
3	143	8	Light irr.
4	161	6	Light irr.
5	147	5	Light irr.
6	154	5	Light irr.
7	149	5	Light irr.
8	140	7	Light irr.
9	150	11	Light irr.
10	144	13	Light irr.
11	150	17	Light irr.
12	149	23	Light irr.

Figure 14: Flooding irrigation Screen

5.2.2.2 Procedure Design

Procedure	Input _irrigation_screen
Purpose	display the input irrigation screen
Calling procedure	transfer_task.input_irrigation_schedule
Called procedure	display,get
Method	**Case** irrigation.schedule_type **append**(List,[1,2,3]). **while**(***not_empty(List))*** **Begin** **get**(Item from List), irr_data(Item,Function,Parameter1,Parameter2) **Case** Item Lable: **display**(Parameter1) Property: **get**(Parmeter1.parameter2) **EndCase** **End***{while}* Irr_data(1,lable,"irrigation_data",_). Irr_data(2,lable,"actual fruit yield",_). Irr_data(37property, current_planting, afy).

procedure	drip_sprinkler_irrigation _screen (monthly bases)
purpose	Display the drip irrigation schedule or sprinkler irrigation schedule.
calling procedure	transfer_task.output_irrigation_schedule
called procedure	display,get
method	**display**("irrigation schedule), display (" Month water quantity interval "), display(" M^3/faddan "), transfer_task.month = 0, **while***(task_parameters.month>= 13)* **Begin** **get**(irrigation_schedule.week_number), **get** (irrigation_ schedule.water_requirement), **get** (irrigation_ schedule.interval), transfer_task.month = transfer_task.month + 1 **End***{while}*

Procedure	flooding _irrigation_Screen (monthly bases)
Purpose	display the flooding irrigation schedule.
Calling procedure	transfer_task.output_irrigation_schedule
Called procedure	display,get
Method	**display** ("irrigation schedule), **display** ("Month water quantity interval irrigation"), **display** (" M^3/faddan type "), transfer_task.month = 0, **while**(***task_parameters.month>= 13)*** **Begin** **get** (irrigation_ schedule.monrh), **get** (irrigation_ schedule.water_requirement), **get** (irrigation_ schedule.interval), **get** (irrigation_ schedule.irrigation_type), transfer_task.month = transfer_task.month + 1 **End***{while}*

5.3 Library of Intelligent Problem Solvers

5.3.1 Mathematical Depenency Mechanism

MDM works on the dependency graph illustrated in sections 4.1.1. The input to this module is just a special node called the root. The calculated values will be attached to each node in the graph. Figure 19 explains the algorithm of this module. As shown in this figure there are five sub-modules which are called from MDM namely: implication_node,look_up_node, mathematic_node, pre_condition_mathematic_node and post_condition_mathematic_node handeling. The goal of the implication_node handling is to execute the relation attached the node to get value. The goal of the look_up_node handling is to execute the table attached the node to get value. The goal of the mathematical-node handling is to execute the equation attached the node to get value. The goal of the pre and post condition_mathematic_node handling are to execute the relation attached the node to get value. The dependency mechanism calls itself recursively to compute the value of this node or to traverse the dependency nodes downward.

```
Dependency Mechanism (root-node)
Begin
   get the Sub-list  of root-node
  While(not empty-list)
       Begin
         get node from list Case node.type

              table                    :  If (node.visited = no) Then
                                             If (not empty sub_list of node) Then
                                                Dependency Mechanism (node)
                                                 Look_up-node handling (node)
                                              Else
                                                 Look_up-node handling (node)
                                              EndIf
                                           EndIf

              relation                 : If (node.visited = no) Then
                                             If (not empty sub_list of node) Then
                                                 Dependency Mechanism (node)
                                                  Implication-node handling(node)
                                             Else
                                                  Implication-node handling(node)
                                              EndIf
                                           EndIf

              function                  : If (node.visited = no) Then
                                             If (not empty sub_list of node) Then
                                                Dependency Mechanism (node)
                                                Mathematical-node handling (node)
                                             Else
                                                Mathematical-node handling (node)
                                             EndIf
                                           EndIf
              Pre_condition_function :   If (node.visited = no) Then
                                             Pre_condition_mathematical_node handling(node)
                                             If (Get the selected_node of node.visited=yes)  Then
                                                 Dependency Mechanism (selected_node)
                                                 node.value = selected_node.value
                                             Else
                                                 node.value = 1
                                             EndIf
                                           EndIf
              Post_condition_function :  If (node.visited = no) Then
                                             Dependency Mechanism (node)
                                             Post_condition_mathematical_node handling(node)
                                           EndIf
              Aggregation               : If (node.visited = no) Then
                                             Dependency Mechanism (node)
                                            node.visited = yes
                                           EndIf
       EndCase
   End{while}
End{dependency mechanism}
```

Figure 19: Dependency Mechanism

Sub modules	Description
Look_up_node _handling	Look_up_node_handling(node) /* The purpose of table node handling is to execute the table name.*/ Begin table = node.table_name node.value = look_up(table) node.visited = yes End{ look_up_node handling}
Implication_node_handling	Implication _node handling(node) /* The purpose of implication node handling is to execute the relation name.*/ Begin relation = node.relation-name node.value = Implication(relation) node.visited = yes End{ Implication _node handling}
Mathematical_node_handling	Mathematical _node handling(node) /* The purpose of mathematical node handling computes the equation.*/ Begin equation= node.equation-name node.value = {calculate the equation for this node} node.visited = yes End{ mathematical _node handling}

Pre_condition_mathematical_node_handling	Implication _node handling(node)
	/* The purpose of pre condition mathematical node handling is to execute the relation name.*/
	Begin
	If (node.visited = no) Then
	relation = node.relation-name
	node.value = Implication(relation)
	node.visited = yes
	EndIf
	End{ Implication _node handling}

Post_condition_mathematical_node_handling	Implication _node handling(node)
	/* The purpose of post condition mathematical node handling is to execute the relation name.*/
	Begin
	If (node.visited = no) Then
	relation = node.relation-name
	node.value = Implication(relation)
	node.visited = yes
	EndIf
	End{ Implication _node handling}

Figure 19: Dependency Mechanism (cont.)

5.3.2 Implication Mechanism

The *implication mechanism* works on an implication relation by satisfying the relation left-hand side and conclude its right-hand side. Figure 20 explains the algorithm of this mechanism.

```
function Implication (Relation) :boolean
Begin
    get the list_of_Rules
    While(not empty-list)
        Begin
            get rule from list
            put the condition part of rule in condition-part-rule-list
            right_part = true
            While(not empty condition-part-rule-list)OR (right-part = false))
                Begin
                    get (object,attribute,opreator,value) from condition-part-rule-list
                    source = Getsource(object,attribute)
                    if (source = user) Then
                        Getvalue_user(real_value)
                        right-part = satisfay_condition(operator,value,real-value)
                    Else if (source = database)
                            Getvalue_db(real_value)
                            right-part = satisfay_condition(operator,value,real-value)
                        Else if (source = implication)
                        Getname(relation)
                        Implication(relation)
                    EndIf
                End{while}
                If right_part = true Then put the rule in the database
    End{while}
    If empty database Then implication = false  Else implication = true
End.
```

Figure 20: Implication Mechanism

5.3.3 Look up Mechanism

The goal of the look_up-node handling is to execute the table name attached the node to get value.

```
look_up(table)
Begin
        Get input_list from table.input
        While (not empty input_list)
        Begin
            select (concept.property from input_list)
            Value = get(concept.property),
            append(table_list, value),
        End{while}
        Get(output_list table.output)
        Contact(table_list,, output_list, table_items)
        table(table_items)
End.{look_up}
```

Figure 21: Look up Mechanism

7. Meta Knowledge and Mechanism Selector

These component are used by the inferences. The objective is to accept data, as input, related to the knowledge representation and produces, as output, one mechanism to be used by the inferences.In our application, we encoded the meta knowledge (see figure 22) as a set of facts to be used by the mechanism selector (see figure 23).

```
domain_model_representation(et0_model, _ , dependency_graph).
domain_model_representation(eta_model, _, dependency_graph).
domain_model_representation(pawc_model,  _ , dependency_graph).
domain_model_representation(interval_model, _, dependency_graph).
domain_model_representation(water_requirement_model,_,dependency_graph).
mechanism(relation, implication_mechanism).
mechanism(function, computional_mechanism).
mechanism(dependency_graph,mathematical_dependency_mechanism).
```

Figure 22: The Meta Knowledge Representation

```
Mechanism_selector(+Domain_model, +Part_identifier, -Mechanism_name)
Begin
    get_representation(Domain_model, Part_identifier, Representation),
    mechanism(Representation, Mechanism_selection),
End
```

Figure 23: Mechanism Selector Algorthim

8. Irrigation Schedule Repository

Irrigation schedule repository is part of working memory of the irrigation schedule. It consists of irrigation schedule and task parameters. This collection is done through special data structure called repository, which receives output resulted form each task as well as the computional model during execution. The following table describes the property of each component.

Repository	Property	facets	
irrigation_schedule	month	integer	1:12
	water_requirement	real	1:1000
	interval	integer	0:7
	irrigation_type	nominal	Light_irrigation, medium_irrigation, heavy_irrigation
task_parameters	month	integer	1:12
	current_month	integer	1:12