# Interoperability and Upgradability Improvement for Context-Aware Systems in Agriculture 4.0

Quang-Duy Nguyen

# UNIVERSITÉ CLERMONT AUVERGNE

**École Doctorale des Sciences Pour l'Ingénieur**

# Doctoral Thesis

---

*Interoperability and Upgradability Improvement for Context-Aware Systems in Agriculture 4.0*

---

*presented by* **Quang-Duy NGUYEN**

*for the degree of* **Doctor of Philosophy** *in* **Computer Science**

*defended on* 16 July 2020

Supervisors:

| | |
|---|---|
| Dr. Jean-Pierre CHANET (TSCF, INRAE) | Thesis Director |
| Dr. Catherine ROUSSEY (TSCF, INRAE) | Advisor |
| Dr. Christophe de VAULX (LIMOS) | Advisor |

Committee Members:

| | |
|---|---|
| Prof. Engelbert MEPHU NGUIFO (LIMOS) | Committee Chair |
| Prof. Gilles BELAUD (SupAgro) | Examiner |
| Prof. Robert LAURINI (Knowledge Systems Institute, USA) | Examiner |
| Dr. Nathalie HERNANDEZ (IRIT) | Rapporteur |
| Prof. Frédérique LAFOREST (LIRIS) | Rapporteur |

*Dedicate to my father in heaven, who taught me always to be hard-working, self-respect, and willing to learn new things.*

*Dedicate to my mother, who raises my soul with love, kindness, and tolerance.*

*Dedicate to the GOD who is never tired of making new challenges to test my courage and to help me to complete myself.*

# Acknowledgments

*"Give me a lever long enough and a fulcrum on which to place it, and I shall move the world."* – **Archimedes**

---

As a rugged person, I felt challenged in expressing my thankfulness to those who have helped me accomplish this great step in my research career. Indeed, I was worried that my words are cold and flat, and these following people deserve more than that.

First of all, I would like to address my most profound appreciation to Dr. Jean-Pierre CHANET, Dr. Catherine ROUSSEY and Dr. Christophe de VAULX. In my eyes, they have always been amazing advisors, patient guides, and great companions throughout my three and a half years doctoral program. Without them, I could not go this far in the research, and this dissertation would not be in your hands today.

I would like to thank all of the committee members: Prof. Engelbert MEPHU NGUIFO, Prof. Gilles BELAUD, Prof. Roberto LAURINI, Dr. Nathalie HERNANDEZ and Prof. Frédérique LAFOREST, for their time and effort in helping me to evaluate and finish this work.

I would like to thank Dr. María POVEDA-VILLALÓN and Mr. Stephan BERNARD, for their contributions to this thesis and the precious knowledge and experience they have shared with me.

I would like to thank Mr. Philippe RAMEAU and Mrs. Laure MOIROUX-ARVIS, for their support in the system deployment and experimentation part.

I would like to thank Dr. Jean CONNIER, for proofreading the french texts in this dissertation.

I would like to thank Dr. Van-Tho NGUYEN, Dr. Sandro BIMONTE, Dr. Tien-Linh LE and Dr. Gil DE SOUSA, for their valuable advice throughout my research.

I would like to thank all of my professors in EDSPI, FLEURA, and SUAPS, UCA, my colleagues in INRAE and LIMOS, and my friends met in France and in Vietnam, for every new lesson, for their loyal support, and for every enjoyment that they have shared with me during my study.

I would like to acknowledge the project ConnecSens, the region Auvergne-Rhône-Alpes, and the Fonds Européen de DÉveloppement Régional (FEDER), for funding my research.

Last but not least, I would like to acknowledge the research center INRAE Clermont-Auvergne-Rhône-Alpes, for hosting me so warmly for the three and a half years of my doctoral program.

# Interoperability and Upgradability Improvement for Context-Aware Systems in Agriculture 4.0


## Abstract

The next evolution of agriculture is Agriculture 4.0. Agriculture 4.0 is about using technologies of the Internet of Things (IoT) and Context-Aware Systems (CASs) to increase the performance of farming activities. A CAS can react automatically and adequately to the environment based on its context. Applying CASs in agriculture can reduce farm labor and increase the precision of farming activities. However, it encounters two challenges specific to agriculture. The first challenge relies on the need to upgrade a CAS regularly with new computing devices or software programs without changing its functionality. Indeed, natural factors, such as violent weather and wild animals, can damage the computing devices located on farmland. Moreover, after each farming season, farmers may need to upgrade their system with new computing devices and software programs. The second challenge is the data heterogeneity generated from a CAS. In agriculture, various phenomena involve the need to have different sensor devices that make numerous types of measurements and produce heterogeneous data. Representing all of these heterogeneous data is necessary for the interoperability of different computing devices in a CAS or the interoperability between different CASs in the IoT ecosystem. This thesis proposes three contributions. The first contribution addresses the first challenge. It is a new architecture based on the microservice mindset that allows system developers to focus on the services' goals rather than the computing devices and software programs of a CAS. This new architecture is called the stack of services for CASs. The second contribution addresses the second challenge. It is a new ontology for CASs named CASO. The ontology provides a vocabulary to model heterogeneous data generated from CASs and embodies a mechanism to make rules for reasoning. The third contribution is to build a decision support system (DSS) for the irrigation CAS in the research unit TSCF, INRAE. The design of the DSS relies on the stack of services for CASs. Moreover, the DSS uses a new ontology called IRRIG, a specialization of CASO for irrigation. The DSS is an automation version of the manual irrigation method IRRINOV®. All the guidelines for farmers in IRRINOV® are transformed into rules for reasoning. The contributions of this thesis are going to be applied to build a smart irrigation CAS deployed in AgroTechnoPôle, located in Montoldre, France.



**Keywords**: Agriculture 4.0, Internet of Things, Context-Aware System, Microservice, Stack of Services, Wireless Sensor Network, Ontology, Rules-based inference, Decision Support System.

# Amélioration de l'Interopérabilité et de l'Évolutivité pour les Systèmes Contextuels dans l'Agriculture 4.0


## Résumé

La prochaine évolution de l'agriculture est l'Agriculture 4.0. Dans ce domaine, les nouvelles technologies de l'Internet des Objets (IdO) et les systèmes contextuels sont utilisés pour améliorer les performances des activités agricoles. Un système contextuel est un système capable de réagir automatiquement et adéquatement en fonction du contexte. Le fait d'utiliser un tel système permet non seulement de réduire la charge de travail des agriculteurs, mais aussi d'améliorer la précision des activités agricoles. Cependant, leur emploi dans le monde rencontre deux obstacles spécifiques. Le premier obstacle est le besoin de mettre régulièrement à jour le système contextuel sans changer sa fonctionnalité. Ce besoin s'appuie sur le fait que l'agriculture est une activité saisonnière, avec un lieu de travail externe, ce qui implique plusieurs facteurs imprévisibles qui influent sur les aspects logiciels et matériels du système. Le deuxième obstacle est l'hétérogénéité de données générées à partir du système contextuel. Dans le domaine agricole, on trouve des capteurs variés observant des phénomènes variés et produisant des données également variées. Représenter ces données est un fait nécessaire pour l'interopérabilité des dispositifs à l'intérieur un système contextuel, ou pour l'interopérabilité de plusieurs systèmes contextuels différents à l'intérieur l'écosystème de l'IdO. Cette thèse propose trois contributions. La première est une architecture s'appuyant sur le principe de microservice. Cette architecture est une pile de services pour les systèmes contextuels, qui permet aux développeurs d'un système de se focaliser sur les objectifs des services plutôt que leurs aspects logiciels et matériels. La deuxième contribution est une ontologie, intitulé CASO, dédiée aux systèmes contextuels. Cette ontologie fournit un vocabulaire pour modéliser les données générées par le système contextuel. De plus, elle inclut un mécanisme pour créer des règles de raisonnement. La troisième contribution est un système d'aide à la décision (SAD) pour l'irrigation automatique, développé à partir d'IRRINOV®, une méthode d'irrigation manuelle. Il fait partie d'un système contextuel dédié à l'irrigation de l'équipe TSCF d'INRAE. Ce SAD est basé sur la pile de services pour les systèmes contextuels, et utilise l'ontologie IRRIG, une spécialisation de CASO dédiée à l'irrigation. Les trois contributions vont être appliquées dans un système contextuel d'irrigation déployé dans l'AgroTechnoPôle, situé à Montoldre, en France.



**Mots-clefs** : Agriculture 4.0, Internet des Objets, Système contextuel, Microservice, Pile de Services, Réseaux de Capteurs Sans Fils, Ontologie, Inférence à base de Règles, Système d'Aide à la Décision.

# Nâng Cao Khả Năng Tương Tác và Nâng Cấp của Hệ Thống Phản Ứng Theo Hoàn Cảnh trong Nông Nghiệp 4.0


## Tóm Tắt

Nông Nghiệp 4.0 sử dụng các công nghệ trong Internet kết nối vạn vật và hệ thống phản ứng theo hoàn cảnh nhằm nâng cao hiệu suất các hoạt động nông nghiệp. Hệ thống phản ứng theo hoàn cảnh là một hệ thống phản ứng một cách tự động và phù hợp dựa trên hoàn cảnh thực tế. Hệ thống này giúp giảm bớt công việc của nông dân cũng như tăng độ chính xác trong các hoạt động nông nghiệp. Tuy nhiên, do tính chất đặc thù của nông nghiệp, việc sử dụng hệ thống này đối mặt với hai khó khăn. Khó khăn thứ nhất là một hệ thống trong nông nghiệp cần được nâng cấp thường xuyên. Thật vậy, các thiết bị trong hệ thống này đặt ngoài trời thường dễ bị ảnh hưởng do tác động bên ngoài, chúng có thể cần được thay thế hoặc nâng cấp. Hơn nữa, nông nghiệp là một hoạt động thời vụ, do đó sau mỗi vụ mùa một hệ thống trong nông nghiệp có thể được nâng cấp theo yêu cầu từ các nhà quản lý nông nghiệp. Khó khăn thứ hai là sự đa dạng của dữ liệu sinh ra trong nông nghiệp. Các hiện tượng trong nông nghiệp rất đa dạng và chúng thường được quan sát bởi các thiết bị cảm biến khác nhau. Các thiết bị cảm biến này sinh ra các loại dữ liệu khác nhau. Việc biểu diễn tất cả dữ liệu nhằm phục vụ cho sự tương tác giữa các thiết bị trong một hệ thống phản ứng theo hoàn cảnh, cũng như giữa các hệ thống riêng biệt, tương đối phức tạp. Luận án này giới thiệu ba đóng góp khoa học. Đóng góp thứ nhất là một mô hình thiết kế cho hệ thống phản ứng theo hoàn cảnh dựa trên các dịch vụ tinh giản. Một hệ thống phản ứng theo hoàn cảnh được thiết kế dựa trên các dịch vụ tinh giản có thể tập trung vào mục đích của các dịch vụ thay vì phần cứng hay phần mềm của hệ thống, qua đó việc nâng cấp hệ thống sẽ bớt phụ thuộc vào thiết bị và chương trình phần mềm. Đóng góp thứ hai là một bản thể học cho hệ thống phản ứng theo hoàn cảnh nói chung với tên gọi CASO. Bản thể học này cho phép biểu diễn tất cả dữ liệu sinh ra từ một hệ thống phản ứng theo hoàn cảnh. Ngoài ra, bản thể học này hàm chứa một cơ chế cho phép người dùng viết các luật cho máy suy luận. Đóng góp thứ ba là một hệ thống ra quyết định tưới tiêu tự động được dịch từ một phương pháp tưới tiêu thủ công cho nông dân với tên gọi IRRINOV®. Hệ thống ra quyết định này sử dụng bản thể học IRRIG, một trường hợp đặc thù phục vụ cho tưới tiêu phát triển từ CASO. Đóng góp từ luận án này được sử dụng thực tiễn trong khu nông nghiệp công nghệ cao có tên AgroTechnoPôle, Montoldre, cộng hòa Pháp.



**Từ khóa**: Nông nghiệp 4.0, Internet kết nối vạn vật, Hệ thống phản ứng theo hoàn cảnh, Dịch vụ tinh giản, Ngăn xếp các dịch vụ tinh giản, Mạng cảm ứng không dây, Bản thể học, Máy suy luận, Hệ thống ra quyết định.

# Contents

**List of Figures** **ix**

**List of Tables** **xi**

**Acronyms** **xiii**

**Glossary** **xv**

**Introduction** **1**

**1 Architecture for Context-Aware Systems Using a Stack of Services** **9**
1.1 Context Life Cycle of Context-Aware Systems . . . . . 10
1.2 Overview of System Design Approaches . . . . . 12
1.3 Stack of Services for Context-Aware Systems . . . . . 13
1.3.1 Services of the Acquisition Phase . . . . . 14
1.3.2 Services of the Modeling Phase . . . . . 15
1.3.3 Services of the Analysis Phase . . . . . 16
1.3.4 Services of the Exploitation Phase . . . . . 16
1.4 Six Smart Farming Systems . . . . . 17
1.4.1 Kirby Smart Farm . . . . . 18
1.4.2 PLANTS . . . . . 21
1.4.3 Phenonet-OpenIoT . . . . . 23
1.4.4 FarmBeats . . . . . 26
1.4.5 BIO-ICT System for Precision Irrigation . . . . . 28
1.4.6 System of Hwang et al. . . . . 30
1.5 Comparison of Six Smart Farming Systems . . . . . 32
1.6 Summary and Discussion . . . . . 34

**2 Context-Aware System Ontology for Data Heterogeneity** **35**
2.1 Ontological Approach for the Interoperability of Data Semantic . . . 36
2.2 Comparison of SOSA/SSN and SAREF . . . . . 39
2.2.1 Requirements Extraction . . . . . 40

2.2.2 Ontology Requirements Coverage . . . . . 41
2.3 CASO: an Ontology Dedicated to Context-Aware Systems . . . . . 43
2.4 Evaluation of CASO . . . . . 46
2.5 Summary and Discussion . . . . . 47

**3 Development of an Irrigation Decision Support System 49**
3.1 Method IRRINOV® . . . . . 50
3.2 Methodology Combined of LOT and Mini-Waterfall . . . . . 53
3.3 System Specification . . . . . 55
3.4 System Design . . . . . 56
3.4.1 Irrigation Ontology for Experimentation in TSCF . . . . . 57
3.4.2 Crop Growth Workflow Data Modeling and Reasoning . . . . 58
3.4.3 Rain Quantity Workflow Data Modeling and Reasoning . . . . 69
3.4.4 Soil Moisture Workflow Data Modeling and Reasoning . . . . 82
3.4.5 Crop Water Need Workflow Data Modeling and Reasoning . . 100
3.4.6 Services of the Irrigation Decision Support System in TSCF . 116
3.5 System Implementation . . . . . 117
3.6 System Testing . . . . . 120
3.6.1 Evaluation Using Unit Test . . . . . 121
3.6.2 Evaluation Using Sample Test . . . . . 124
3.6.3 Evaluation Using System Test . . . . . 128
3.7 Summary and Discussion . . . . . 131

**Conclusion 133**

**References 137**

**Appendix A New Vocabularies Defined in CASO and IRRIG 145**

**Appendix B 14-Pages Summary in French 171**

**Vita 185**

# List of Figures


1 The mindset of IoT . . . 3
2 IoT technologies . . . 3
3 Prototype of a CAS in e-agriculture . . . 4
4 The practice in traditional irrigation . . . 5
5 The practice of using a CAS in modern irrigation . . . 5
6 Context life cycle of a smart farming CAS . . . 11
7 Stack of services for CASs . . . 14
8 Description of the Kirby Smart Farm system . . . 20
9 Description of the PLANTS system . . . 22
10 Description of the Phenonet-OpenIoT system . . . 25
11 Description of the FarmBeats system . . . 27
12 Description of the precision irrigation use case of the BIO-ICT system . . . 29
13 Description of the system of Hwang et al. . . . 31
14 Stack of interoperability for CASs . . . 37
15 Overview of CASO . . . 45
16 Evaluation report of CASO using OPPS! . . . 46
17 Issue tracker interface of the CASO GitHub repository . . . 47
18 Prototype of the IRRINOV® station . . . 51
19 Procedure of the LOT methodology . . . 54
20 Procedure of the mini-waterfall methodology . . . 54
21 Mini-Waterfall combined with LOT methodology to develop a system . . . 55
22 Screenshot of a part of the competency questions document containing the questions and answers from CQ1.1 to CQ1.4 . . . 56
23 Four workflows of the CAS smart irrigation . . . 57
24 Overview of IRRIG . . . 58
25 CropGrowth observation on 27/07/2013 . . . 61
26 States of the property CropGrowth . . . 62
27 Automata of the CropGrowth states and their transitions . . . 63
28 CropGrowth deduction on 14/08/2013 (part 1) . . . 64
29 CropGrowth deduction on 14/08/2013 (part 2) . . . 65
30 Time scales of the crop growth workflow . . . 66

31 RainQuantity observation in [07:00:00, 08:00:00[ on 14/08/2013 . . 70
32 RainDailyTotalQuantity observation on 14/08/2013 . . . . . 73
33 DelayDuration observation on 14/08/2013 . . . . . 75
34 Thresholds of the states of the property RainIntensity . . . . . 77
35 Automata of RainIntensity states and their transitions . . . . . 78
36 RainIntensity deduction on 14/08/2013 (part 1) . . . . . 79
37 RainIntensity deduction on 14/08/2013 (part 2) . . . . . 80
38 States of the property RainIntensity . . . . . 81
39 Time scales of the rain intensity workflow . . . . . 82
40 Soil30cmDepthMoisture observation at 08:00:00 on 14/08/2013 . . . . . 85
41 Soil30cmDepthDailyAverageMoisture observation on 14/08/2013 . . . . . 87
42 RootZoneDailyAverageMoisture observation on 14/08/2013 . . . . . 90
43 Thresholds of the states of the property RootZoneMoistureLevel . . . . . 93
44 Automata of the RootZoneMoistureLevel states and their transitions . . 94
45 RootZoneMoistureLevel deduction on 14/08/2013 (part 1) . . . . . 96
46 RootZoneMoistureLevel deduction on 14/08/2013 (part 2) . . . . . 96
47 States of the property RootZoneMoistureLevel . . . . . 97
48 Time scales of the soil moisture workflow . . . . . 98
49 Automata of CropWaterNeed states and their transitions . . . . . 103
50 CropWaterNeed deduction on 14/08/2013 (part 1) . . . . . 106
51 CropWaterNeed deduction on 14/08/2013 (part 2) . . . . . 107
52 States of the property CropWaterNeed . . . . . 108
53 Time scales of the crop water need workflow . . . . . 109
54 SleepingDuration observation on 14/08/2013 . . . . . 115
55 Services run by the irrigation DSS in TSCF . . . . . 117
56 Algorithm in Ontogen for the DSS . . . . . 119
57 Devices of the CAS irrigation system in TechnoHall . . . . . 134
58 AgroTechnoPôle in Montoldre, France . . . . . 135
59 Un prototype de système contextuel pour l'irrigation . . . . . 172
60 Le cycle de fonctionnement d'un système contextuel dédié à l'irrigation . 174
61 La pile de services pour les systèmes contextuels . . . . . 175
62 L'ontologie CASO . . . . . 178
63 La méthodologie combinée de midi-cascade et LOT . . . . . 181
64 Quatre flux opérationnels du système contextuel dédié à l'irrigation . . . 182
65 L'ontologie IRRIG . . . . . 183

# List of Tables


1 Services found in the six smart farming systems . . . . . 33
2 Requirements coverage by SOSA/SSN and SAREF . . . . . 42
3 List of prefixes for CASO and IRRIG . . . . . 44
4 Decision table of the relation of the watering cycle duration and the soil moisture for maize at growth stage V2 . . . . . 52
5 Observations of crop growth stage stored in the Arvalis dataset . . . . . 59
6 Rule for the property CropGrowth to reach the R5 state . . . . . 67
7 Rule for the property CropGrowth to reach the V7 state . . . . . 68
8 Function getRainDailyTotalQuantity(p,d) . . . . . 72
9 Function getDelay(d) . . . . . 74
10 Rule for the property RainIntensity to reach a state . . . . . 83
11 Function getDailyAggregatedMoisture(p,d) . . . . . 86
12 Function getRootZoneDailyAverageMoisture(d) . . . . . 89
13 Rule for the property RootZoneMoistureLevel to reach a state . . . . . 99
14 Conditions of CropWaterNeed state transitions . . . . . 104
15 Rule for the property CropWaterNeed to reach the NonApplicable state . 110
16 Rule for the property CropWaterNeed to reach the No state (part 1) . . 111
17 Rule for the property CropWaterNeed to reach the Yes state (part 1) . . 112
18 Rule for the property CropWaterNeed to reach the Unavailable state . . 113
19 Function getSleeping(d) . . . . . 114
20 Unit tests of rules for the state of CropGrowth deduction . . . . . 122
21 Unit tests of rules for the state of RainIntensity deduction . . . . . 123
22 Unit tests of rules for the state of RootZoneMoistureLevel deduction . . 124
23 Unit tests of rules for the state of CropWaterNeed deduction (part 1) . . 125
24 Unit tests of rules for the state of CropWaterNeed deduction (part 2) . . 126
25 Sample tests of rules for the state of CropGrowth deduction . . . . . 127
26 Sample tests of rules for the state of RainIntensity deduction . . . . . 127
27 Sample tests of rules for the state of RootZoneMoistureLevel deduction . 128
28 Experiment during the period from 22/07/2013 to 31/07/2013. . . . . 130
29 Experiment during the period from 31/07/2013 to 14/08/2013. . . . . 131
30 Experiment during the period from 14/08/2013 to 21/08/2013. . . . . 131

31 Experiment during the period from 21/08/2013 to 03/09/2013. . . . . 132
32 Rule for the property CropGrowth to reach the V7d20 state . . . . . 159
33 Rule for the property CropGrowth to reach the R1 state . . . . . 160
34 Rule for the property CropGrowth to reach the R1d15 state . . . . . 160
35 Rule for the property CropGrowth to reach the R5hg45 state . . . . . 161
36 Rule for the property CropWaterNeed to reach the No state (part 2) . . 162
37 Rule for the property CropWaterNeed to reach the No state (part 3) . . 162
38 Rule for the property CropWaterNeed to reach the No state (part 4) . . 163
39 Rule for the property CropWaterNeed to reach the No state (part 5) . . 164
40 Rule for the property CropWaterNeed to reach the Yes state (part 2) . . 164
41 Rule for the property CropWaterNeed to reach the Yes state (part 3) . . 165
42 Rule for the property CropWaterNeed to reach the Yes state (part 4) . . 166
43 Rule for the property CropWaterNeed to reach the Yes state (part 5) . . 166
44 Rule for the property SleepingDuration to get a value (part 1) . . . . . 167
45 Rule for the property SleepingDuration to get a value (part 2) . . . . . 168
46 Rule for the property SleepingDuration to get a value (part 3) . . . . . 168
47 Rule for the property SleepingDuration to get a value (part 4) . . . . . 169
48 Rule for automatically inferring GreaterThan relations . . . . . 170
49 Rule for automatically inferring LesserThan relations . . . . . 170
50 Services disponibles dans les six systèmes agricoles intelligents . . . . . 177

# Acronyms

**ACAS** Adaptive Context-Aware System

**API** Application Protocol Interface

**CAS** Context-Aware System

**CASO** Context-Aware System Ontology

**CEP** Complex Event Processing

**DSS** Decision Support System

**e-Agriculture** Agriculture using Information and Communication Technologies

**ETSI** European Telecommunication Standardization Institute

**GDU** Growing Degree Unit

**GPS** Global Positioning System

**GSN** Global Sensor Network

**ICT** Information and Communication Technologies

**IoT** Internet of Things

**IRRIG** Ontology specified for IRRIGation Systems using the method IRRINOV®

**LOT** Linked Open Terms

**LSM** Linked Stream Middleware

**OGC** Open Geospatial Consortium

**RDF** Resource Description Framework

**RFID** Radio Frequency IDentification

**SAREF** Smart Appliances REFerence Ontology

**SDUM** Scheduler and the Service Delivery and Utility Manager

**SOSA** Sensor-Observation-Sampling-Actuator Ontology

**SRWL** Semantic Web Rule Language

**SSN** Semantic Sensor Network

**W3C** World Wide Web Consortium

**WSN** Wireless Sensor Network

**X-GSN** Extension-GSN

# Glossary

**Adaptive Context-Aware System** Context aware system that can modify its behaviour according to changes in the application's context *(Efstratiou, 2004)*.

**Aggregate Data** Data get from an aggregation. Aggregation is a computation of the function of a computer program.

**Clean Data** the data that is processed so that it can be used correctly by one or several applications of an information system. The data before the processes to become clean data is raw data.

**Cloud Computing** All data must be uploaded to centralized servers, and after computation, the results need to be sent back to devices *(Yu et al., 2018)*.

**Computation** An action of calculation realized by an electronic device. Three computation types considered in this dissertation are measurement, aggregation, and deduction.

**Computing Device** An electronic device with processing and storage capabilities, for example, a computer, a mobile phone, and an embedded device.

**Context** Any information that can be used to characterise the situation of an entity *(Efstratiou, 2004)*. Context of an information system is a set of entities characterised by their state, plus all information that can help to derive any state changes of these entities *(Sun et al., 2016)*.

**Context-Aware System** Uses context to provide relevant information and services to users, where relevance depends on the user's task *(Abowd et al., 1999)*.

**Decision Support System** An application of an information system or an information system that specialized in provides information to suggest farmers in their activities.

**Deducted Data** Data get from a deduction. A deduction is a process of providing output result of an inference engine.

**Domesticated Species** Domesticated animals and plants produces agriculture products and be affected by farming activities *(Renting et al., 2009)*.

**Embedded Device** A type of computing device. In this dissertation, an embedded device means a sensor device and an actuator device. A sensor device is equipped with sensors and a transceiver module to monitor phenomena *(Deepika and Rajapirian, 2016)*. An actuator device with communication capability can receive the control information to react to the environment.

**Hardware Component** A term used in the context-aware system. It can be a computing device or a part of a computing device belonging to the system.

**High-Level Context** A type of context of information systems that contain qualitative data *(Sun et al., 2016)* All the data contributing to the high-level context of a system is also called high-level context data.

**Information System** System that first collects, stores and processes data to transform them into information. Then, the information is further processed and turns into another form satisfying the applications of the system.

**Internet of Things** Allows people and things to be connected Anytime, Anyplace, with Anything and Anyone, ideally using Any path/network and Any services *(International Telecommunication Union, 2005)*.

**Interoperability** A characteristic of a product or system, whose interfaces are completely understood, to work with other products or systems, present or future, in either implementation or access, without any restrictions, as defined by the Association Francophone des Utilisateurs de Logiciels Libres (AFUL).

**Low-Level Context** A type of context of information systems that contain quantitative data *(Sun et al., 2016)* All the data contributing to the low-level context of a system is also called low-level context data.

**Measured Data** Data get from a measurement. Measurement is the data generation of a sensor device.

**Microservice** A style of engineering highly automated, evolvable software systems made up of capability-aligned microservices *(Nadareishvili et al., 2016)*. Each microservice is an independently deployable component of limited scope that supports interoperability through message-based communication. They are small, highly decoupled and focus on doing a small task *(Newman, 2015)*.

**Ontological Engineering** The set of activities that concern the ontology development process, the ontology life cycle, the methods and methodologies for building ontologies, and the tool suites and languages that support them *(Gómez-Pérez et al., 2004)*.

**Ontology** A formal, an explicit specification of a shared conceptualization *(Studer et al., 1998)*.

**Precision Agriculture** Farming activities with the mind-set about doing the right thing at the right place at the right time *(Wolfert et al., 2014)*.

**Process** Type of composition whose elements are composed into a sequence or flow of activities and interactions with the objective of carrying out certain work *(ISO/IEC, 2016)*. To put it simply, process is composed of a set of activities to accomplish a goal.

**Semantic Web** A web environment in which information is well-defined meaning, better-enabling computers and people to work in cooperation *(Berners-Lee et al., 2001)*.

**Service** Logical representation of a set of activities that have specified outcomes, is self-contained, may be composed of other services, and is a black box to consumers of the service *(ISO/IEC, 2016)*.

**Smart Farming** Real-time data gathering, processing, and analysis, as well as automation technologies on the farming procedures, allowing the improvement of the overall farming operations and management and more informed decision-making by the farmers *(Kamilaris et al., 2016)*.

**Software Component** A term used in the context-aware system. It can be a software program or a part of a software program belonging to the system.

**Stack of Services for Context-Aware Systems** An architecture inspired by the microservice mindset. It allows system designers to focus on the goal of services. The stack of services for context-aware systems is composed of several services organized in four phases of the context life cycle of context-aware systems.

**State** A qualitative data which changes over time, summarizing a set of information of an entity *(Bendadouche et al., 2012)*. In an information system, the entity and its states are defined for a specific application.

**Upgradability** The ability to replace computing devices and software programs of a system then brings the system up to date or improve its characteristics.

**Watering Cycle** An irrigation system performs a watering activity on all the plots engaged in the system *(Nguyen et al., 2020b)*. Watering cycle duration is the number of days between two consecutive waterings on a same plot.

**Wireless Sensor Network** A network of devices, denoted as nodes, which can sense the environment and communicate the information gathered from the monitored field through wireless links *(Buratti et al., 2009)*.

# Introduction

*"Knowing yourself is the beginning of all wisdom."* – **Aristotle**

Agriculture has always been one of the most vital factors of human civilizations *(Harari, 2015)*. It refers to the science and art of cultivating domesticated species *(International Labour Office, 2011)*. In which, domesticated species are animals and plants that produce agricultural products like milk, fruits, and cotton. These animals and plants are affected by farming activities such as sowing, monitoring, and irrigation. Thanks to agriculture, humans can satisfy their basic physiological needs of food and clothing *(Maslow, 1943)*. However, this fact is no longer self-evident in a short time since agriculture is facing two critical challenges. First, the population explosion and the scarcity of natural resources are followed by a new situation that the demand for agricultural products exceeds the production capacity of agriculture *(Sundmaeker et al., 2016)*. Second, climate changes in air temperature, atmospheric carbon dioxide, and the frequency of extreme weather, have negative impacts on agriculture: the reduction of the quality and quantity of agriculture products, the risk of exotic pests, and the difficulties faced by farmers in deploying farming activities *(Urruty et al., 2016)*. Addressing these two challenges, humans have been applying information and communication technologies (ICTs) in agriculture. This approach improves the performance of farming activities; therefore, it increases agricultural productivity and yield, decreases the environmental footprint of agriculture, and replaces farmers' labor by automation. The domain of agriculture using ICTs is also known as e-agriculture.

The development of ICTs by time has a significant impact on the evolution of e-agriculture. At the end of the 20th century, the exponential growth of computing devices and software programs, and the popularity of the global positioning system (GPS) and the wireless sensor network (WSN), are the base for a new evolution in e-agriculture, known as Agriculture 3.0[1]. The aim of Agriculture 3.0 is to support

[1]Agriculture 3.0, together with Agriculture 1.0, 2.0, and 4.0, are a subset of technological

precision agriculture. Precision agriculture is *"about doing the right thing at the right place at the right time" (Wolfert et al., 2014)*; so that, domesticated species can receive the exact treatment that they need *(Netherlands Study Centre for Technology Trends, 2016)*. The early 21st century has been witnessing the rise of the Internet of Things (IoT), which is the base for Agriculture 4.0. In detail, the IoT is a mindset where *"people and things are connected anytime, anyplace, with anything and anyone, ideally using any path/network and any services" (International Telecommunication Union, 2005)*. All of the ICTs that contribute to this mindset are IoT technologies. They can be categorized into three groups: things-oriented vision, internet-oriented vision, and semantic-oriented vision *(Sundmaeker et al., 2010)*. The things-oriented vision includes technologies in building computing devices, such as RFID, NFC tag, and embedded devices. The Internet-oriented vision provides communication standards such as Wifi, Zigbee, LoRa, 4G, and 5G. The semantic-oriented vision is dedicated to resolves the data heterogeneity issues in IoT by using semantic technologies such as ontology. Figure 1 illustrates the mindset of IoT and Figure 2 shows the three groups IoT technologies. Agriculture 4.0 is the next evolution of Agriculture 3.0 in both technologies and vision. Technically speaking, it inherits the technologies available in Agriculture 3.0, also adopting IoT technologies. Its vision has two folds. First, it focuses more on smart farming instead of precision agriculture. Smart farming refers to *"real-time data gathering, processing, and analysis, as well as automation technologies on the farming procedures, allowing the improvement of the overall farming operations and management and more informed decision-making by the farmers" (Kamilaris et al., 2016)*. Second, it insists on sharing data between different systems in one domain or in the IoT cross-domains ecosystem. The IoT cross-domain ecosystem includes many domains such as smart farming, smart home, smart cities, and smart transportation *(Al-Fuqaha et al., 2015)*.

A context-aware system (CAS) is the candidate to undertake smart farming. A CAS is a system that *"uses context to provide relevant information and services to users, where relevant depends on the users' task" (Abowd et al., 1999)*. It can react quickly and adequately to the context changes. Technically speaking, a CAS is composed of connected computing devices. A computing device is an electronic equipment with processing and storage capabilities, such as a computer, a mobile phone, or an embedded device. An embedded device in this dissertation means a sensor device or an actuator device. A sensor device is equipped with a transceiver module and sensors, to monitor phenomena *(Deepika and Rajapirian, 2016)*. An actuator device

revolutions. A technological revolution is a period in which the emergence of a new group of technologies, for example, in mechanics, biology, and ICTs, profoundly changes the way humans work in a specific domain, such as industry or agriculture. The history records four technological revolutions known as technological revolution 1.0, 2.0, 3.0, and 4.0.

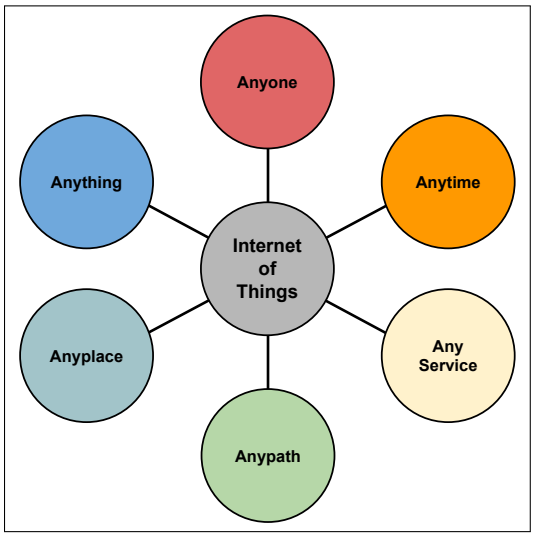



**Figure 1**: The mindset of IoT

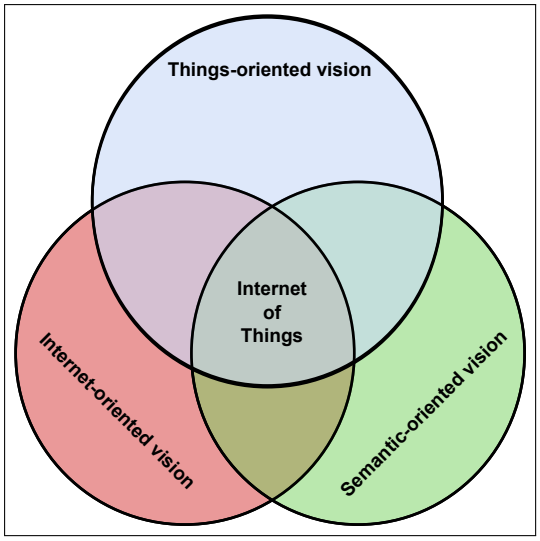



**Figure 2**: IoT technologies

with communication capability can receive control signals to act on the environment. In a CAS, one or several computers work as an information system. An information system can collect, store, and process data to transform them into information. Then, the information is further processed and turned into another form, satisfying some applications of the information system. A typical CAS often comprises three elements: a decision support system (DSS), a WSN, and actuator devices. A DSS is a type of information system specializing in providing information to suggest farmers in their activities. A WSN includes sensor devices that monitor phenomena in the field and send them to the information system. Actuator devices react to the environment under the control of the information system. A modern CAS must be able to connect to the Internet, then can have a cloud computing centralized server. Cloud computing is a concept that a centralized server somewhere in the Internet stores and processes data before sending the processed results back to the computing devices in the local network *(Yu et al., 2018)*. The computing devices in the local network connect to the Internet through a gateway. A gateway is a computing device that plays a role as the data forwarder between different networks. Figure 3 illustrates the prototype of a CAS in e-agriculture.

The following example shows the advantage of e-agriculture compared to agriculture without ICTs. In traditional irrigation, farmers go to their fields to examine the crop development stage and to read the soil moisture measurement provided by the probes[2] put in the soil. They use practical experience or follow an irrigation method to estimate the water needs of the crops manually. Based on

[2]A probe is a soil sensor device in the shape of a tube.

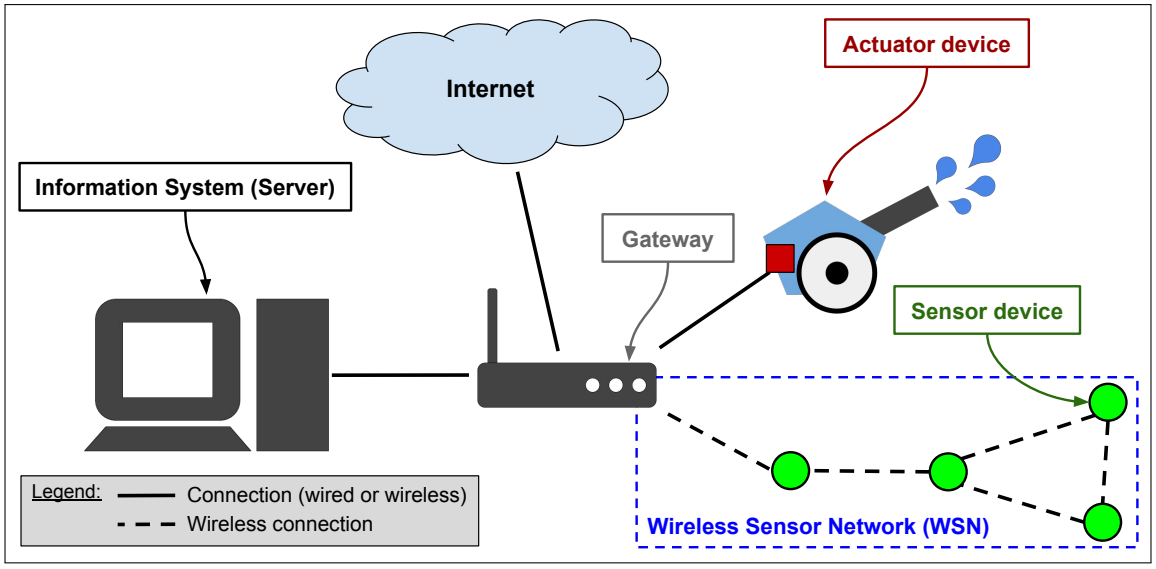



**Figure 3**: Prototype of a CAS in e-agriculture

the estimates, the farmers decide whether to irrigate their fields. Figure 4 illustrates the practice in traditional agriculture. This traditional approach has two significant drawbacks. First, it requires daily observations, often made by farmers. Manual observations are unstable since farmers can make mistakes. Also, they depend strictly on the availability of farmers. Second, the resource shortage problem demands farmers to use water sparingly. Using a CAS can overcome the two above drawbacks. Addressing the first drawback, the WSN can automatically monitor the crops precisely 24/7, as a result, reduce the labor's farmers. Also, the results returned from the WSN are highly accurate. Addressing the second drawback, the DSS uses the observed data to calculate and deduct watering suggestions. A suggestion could be a precise amount of water or a precise time for watering the fields, depending on the irrigation method. Figure 5 illustrates an example of using a CAS in modern irrigation.

The above example shows that the use of CASs, together with ICTs in irrigation, is better than the traditional approach in many senses. It is also true in other farming activities. Besides the advantages, a CAS itself encounters the two following challenges specialized in agriculture.

- Challenge in the upgradability of CASs: Agricultural workplaces are mostly outdoor. Many factors can break outdoor computing devices. For example, random lightning strikes and breaks a soil sensor device. Consequently, broken computing devices need to be replaced by new ones. Ideally, the substitute computing device should have the same model and use the same software program as the broken one. In reality, these conditions might not be satisfied. Thus, it is uneasy for system developers to guarantee the functionality of the

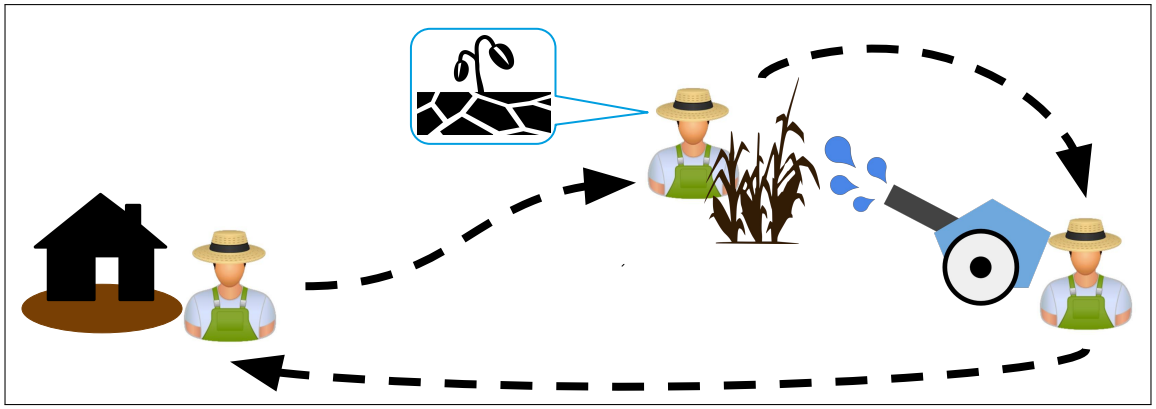


**Figure 4**: The practice in traditional irrigation

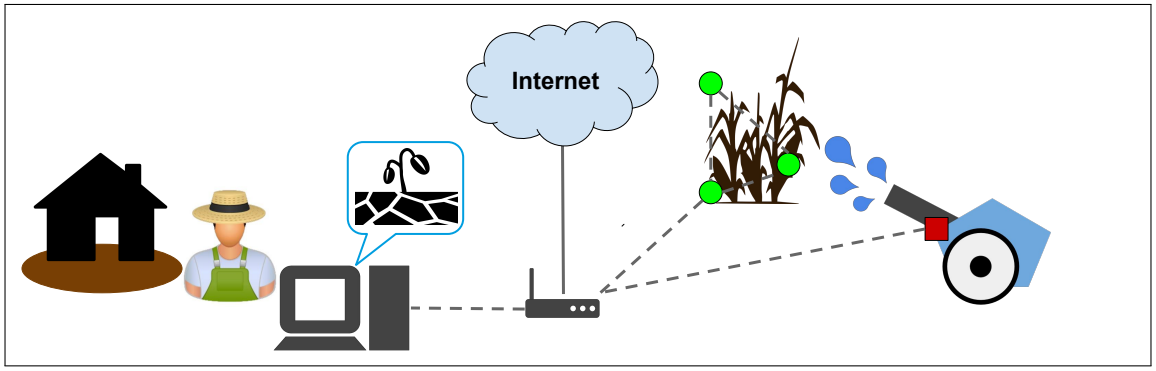



**Figure 5**: The practice of using a CAS in modern irrigation

CAS. Another reason for the need to upgrade a CAS relies on a feature of agriculture: it is a seasonal activity that relates to cycles of food growing and harvesting. After each farming season, farmers may have several adjustments preparing for the next season. One of the changes is to upgrade their CAS with a new version of computing devices and software programs. It raises the same issue: how to upgrade a CAS with new computing devices and software programs without changing its functionality.

- Challenge in the interoperability of CASs: The phenomena needed to be observed in agriculture varies. Consequently, a CAS must have different sensor devices that make different measurements and produce different measured data. For example, a soil sensor makes a soil moisture measurement that produces the soil moisture data of a plot at the moment. A pluviometer makes a rain quantity measurement that produces the rain quantity data of a period. It is necessary to represent all of these heterogeneous data in a CAS to guarantee the interoperability between the computing devices of the CAS. Moreover, agriculture 4.0 also considers the interoperability between CASs in e-agriculture

and the interoperability between CASs in the IoT ecosystem. Representing the data generated by a CAS for its computing devices is uneasy; even worse, representing the data generated by a CAS for other CASs in the cross-domain ecosystem is a significant challenge.

Note that upgradability and interoperability are two qualities of systems; thus, overcoming the two above challenges can improve the qualities of CASs. Two following hypotheses promise to overcome the two above challenges:

- Hypothesis about a system design for CASs independent of computing devices and software programs: It addresses the challenge of the upgradability of CASs. Suppose that there is no delay on the Internet, and all the exchanged messages between computing devices always arrive on time. If there is a system design approach that divides a CAS into logical blocks and focuses on the goal of each logical block regardless of location and type of computing devices and software programs, system developers can freely upgrade the CAS with new computing devices and software programs as long as they satisfy the goal of logical blocks.
- Hypothesis about a sharable data model for CASs: It addresses the challenge of the interoperability of CASs. Suppose that every computing device of every CAS can extract data from received messages without faults. If CASs share the same data model and vocabulary, and every received data goes along with metadata to explain itself, CASs can automatically process these data.

This research aims to overcome the two challenges in the upgradability and the interoperability of CASs. As the final result, there are three contributions to this research as follows.

- Contribution about using a microservice architecture specialized for CASs: This contribution follows the hypothesis of a system design for CASs independent of computing devices and software programs. It presents a new architecture called the stack of services for CASs. This stack of services is composed of a set of logical blocks called services. Each service is a process of a CAS.
- Contribution about using an ontology specialized for CASs: This contribution follows the hypothesis of a sharable data model for CASs. It presents a new ontology called CASO. This ontology contains the fundamental concepts and properties for CASs in general. Consequently, every CAS can use this ontology to model its data. Also, the vocabulary of this ontology implies a mechanism supporting to make generic rules for reasoning.
- Contribution to translating a manual irrigation method into rules: This contribution presents the irrigation DSS in TSCF that uses a new ontology called IRRIG. This ontology is a specialization of CASO dedicated to irrigation.

It is not only for data modeling but also for reasoning. In detail, the rules in the ontology are transformed from the irrigation guideline for farmers in an manual irrigation method named IRRINOV®.

The two first contributions can improve the upgradability and the interoperability of CASs. As a side effect, they can help other system developers to accelerate their development. The third contribution is a specialization of using the two first contributions. Also, it has a meaning in sharing the development experience of developing an expert system that automatizes the irrigation method IRRINOV®. An expert system is a type of DSS that emulates a human expert's decision-making ability by reasoning through a knowledge base *(Jackson, 1999)*. This expert system is going to be a part of the irrigation CAS in the experiment farm in Agrotechnopôle. For more information, Agrotechnopôle is an ecosystem for agricultural machines and digital information systems developed by an alliance of organizations and institutes in Europe. Two members of the alliance dedicated to developing AgrotechnoPôle are INRAE[3] and LIMOS[4].

Note that the contributions and experimentations of this research are in the close system of TSCF. Thus, there is no problem with security, and all the computing devices are available to access. In the real context of the IoT, different systems may have their constraints on security.

This dissertation aims to present the three above contributions. The early part of this introduction has already introduced the most basic information to understand the context and the motivation of this research. The rest of this dissertation is organized as follows.

Chapter 1 addresses the first contribution. At first, it explains the context life-cycle of CASs. Next, it introduces different system design approaches before selecting one for CASs in the IoT. One key section of this chapter is to present the stack of services for CASs as the chosen system design approach. The stack of services for CASs serves as a tool to analyze six smart farming systems to prove that it can cover several CASs.

Chapter 2 focuses on the second contribution. At the beginning, it presents ontology as a solution to overcome the data heterogeneity in IoT. Then, it compares two ontologies SOSA/SSN and SAREF based on the requirements of a CAS use case. This comparison is necessary since one amongst them will be the core of CASO. Finally, this chapter presents the evaluation of CASO.

[3]INRAE is a France's new National Research Institute for Agriculture, Food and Environment, formed by the fusion of INRA and IRSTEA in January 2020.

[4]LIMOS is the Laboratory of Informatics, Modelling, and Optimization of the Systems (UMR 6158), in Aubière, France.

Chapter 3 presents the third contribution. First, it gives the essential information about the irrigation method IRRINOV®. Second, this chapter introduces a new methodology to develop the expert system for daily watering suggestions based on the method IRRINOV®. This chapter's main contribution is the experiences to develop this expert system presented in the four next sections: specification, modeling, implementation, and testing.

Finally, the conclusion sums up the dissertation and then discusses the limitations and difficulties of this research work. Moreover, it opens some perspectives to improve and continue this research in the future.

# Chapter 1

# Architecture for Context-Aware Systems Using a Stack of Services

*"It is not the strongest of the species that survives, nor the most intelligent that survives. It is the one that is the most adaptable to change."* – **Charles Darwin** (as quoted by **Leon C. Megginson**)

---

CAS is a kind of system suitable for unstable environments that require quick reactions against context changes. Agriculture, a domain with workplaces are mostly outdoors, is a typical example of such an environment. In agriculture, domesticated species, including crops and livestock, are easily affected by many natural changeable factors such as air temperature and rain.

CASs in the IoT have two critical characteristics. First, the system can be highly distributed. Second, the hardware and software components of this system can be replaced very fast by time. They involve a need to have a design approach that focuses more on the goal of the components instead of the type of hardware, software, or location. This chapter presents a new design approach called the stack of services for CASs that satisfy the above requirements.

This chapter is organized as follows. Section 1.1 gives more information about CASs: the definition of context and the context life-cycle in CASs. Section 1.2 presents different approaches to design a system and highlights the reason to propose the stack of services for CAS. The main contribution of this chapter is Section 1.3: it defines the stack of services for CASs in detail. Next, Section 1.4 is a state of the art of six smart farming systems based on the stack of services for CASs. Section 1.5 compares the six systems and distinguishes essential and optional services. Finally, Section 1.6 sums up the chapter and opens a discussion.

## 1.1 Context Life Cycle of Context-Aware Systems

Context is the most fundamental element of a CAS. In general, context refers to as *"any information that can be used to characterize the situation of an entity" (Abowd et al., 1999)*. And an entity can be *"a person, a place, or an object that is considered relevant to the interaction between a user and an application, including the user and applications themselves."* However, a context should be specialized for an information system as *"a set of entities characterized by their state, plus all information that can help to derive any state changes of these entities" (Sun et al., 2016)*. Of which, a state is *"a qualitative data which changes over time, summarizing a set of information of an entity" (Bendadouche et al., 2012)*. In an information system, the entity and its states are defined for a specific application. Two types of context are: low-level context and high-level context. The low-level context contains quantitative data, and the high-level context contains qualitative data. Quantitative data can be the binary number, a digital picture, or other kinds of resources that are related to an entity but unable to describe the state of the entity. For example, the number `30` in the context `30 degree C` is related to the `room's temperature`. Qualitative data can be everything that can describe the state of an entity such as `hot` is a state of the `room's temperature`.

The following example supports to understand better the terms of context and entity in an information system. Given that there is an application that receives measured data from some tensiometers to decide whether a plant needs water. In this application, the **plant** is an entity related directly to the goal of this application. It has two states `needs water` and `does not need water`. The **soil** is another entity that relates indirectly to the application's goal. After one measurement of the tensiometer, the data `soil moisture is at 150 cbar` is generated. It belongs to the low-level context. The low-level context transforms into the high-level context of the application after a reasoning process. The high-level context contains two new data. The first data is deducted from the data in the low-level context. This data is represented as `soil is dry`, of which `dry` is the state of the entity **soil**. Second, another data related to the entity **plant** is deducted from the high-level context data related to the entity **soil**. It can be presented as `plant needs water`.

Figure 6 shows the context life cycle of a smart farming CAS. The working procedures of CAS include four phases: acquisition, modeling, analysis, and exploitation. Each phase contains a group of processes that enable the system, at first, to receive input data, process it, then to produce new output data. The figure describes transferred messages between phases and between each phase in the CAS (red box) and the other objects outside this system. The messages between phases in the system are data, low-level context, and high-level context. The messages between

a phase in the system and an object external of the system (external systems, external data sources, human) can be data or information. The description of the four phases of a CAS is as follows.

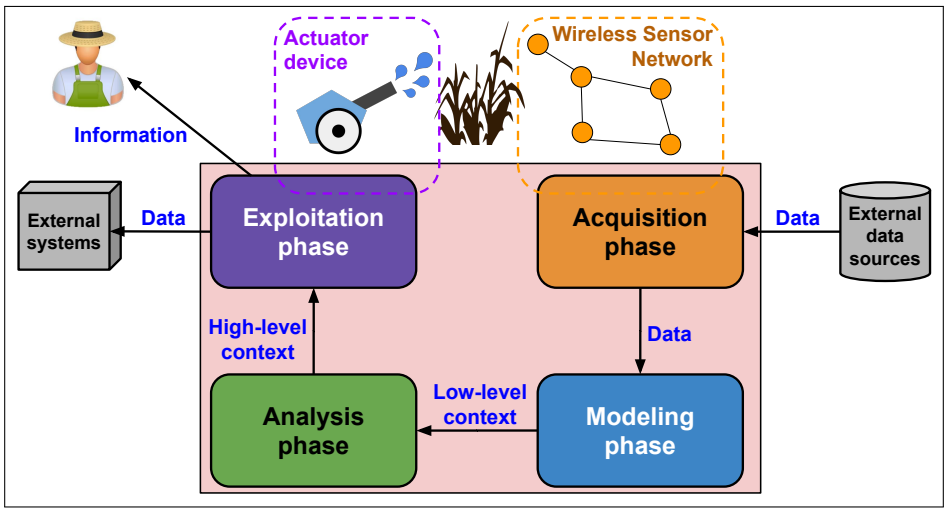



**Figure 6**: Context life cycle of a smart farming CAS

- **Acquisition phase**: A phase contains processes that focus on how the system retrieves measured data directly from the WSN or data collected from external data sources. An external data source could be any source in the Internet that does not belong to the local network of the CAS. This paper defines data sources located inside the system, such as a sensor device of a WSN, as internal sources. The output of this phase is clean data. Clean data refers to the data which are processed and can be used correctly by one or several applications of an information system.
- **Modeling phase**: A phase contains processes that focus on how to model and organize clean data into the system. This phase demands the system equipped with a data model and storage in advance. The input of this phase is clean data derived from the acquisition phase. The system represents and stores the input data, and guarantee that the data becomes meaningful with the system. The output of this phase is the low-level context.
- **Analysis phase**: A phase contains processes that focus on how to transform and enrich the low-level context into the high-level context. The transformation and enrichment are realized by processes such as reasoning, fusion, and aggregation. The input of this phase is the low-level context from the modeling phase. The output of this phase is the high-level context required by the applications of the system.

- **Exploitation phase**: A phase contains processes that focus on how the system uses the high-level context retrieved from the analysis phase to run applications. Three categories of application are user-oriented, external system distribution, and action. According to the applications, this phase's output could be information, data, or physical action realized by an actuator device.

## 1.2 Overview of System Design Approaches

System design is an essential step to develop a system. A system design approach is a prototype to design systems. System developers choose a system design approach appropriate to their goal and the conditions of the expected system. Depending on the complexity and detail of the prototype, it is possibles to have four categories:

- **Design mindset**: Comprising only the principle and ideas to design a system. System developers can use freely to choose and organize the hardware and software components as long as it satisfies the principles of the design mindset. Some examples of design mindsets are microservice and service-oriented architecture (SOA).
- **Architecture**: Comprising several logical blocks, of which, each block has its functions. In general, these blocks follow an order or an organization defined by the architecture. System developers are free to choose hardware and software components that satisfy the functions of blocks. Some examples of architectures are MAPE-K, IoT-A, and Avatar.
- **Platform**: Comprising several suggestions of hardware and software components to form a system. It usually is a blueprint detail the type of hardware and software components that system developers need to follow. Some examples of platforms are Agri-IoT, FIWARE, DIMMER, and CityPulse.
- **System**: Comprising the detail hardware and software components and their configuration. System developers normally run and maintain a system in real experimentation. Some examples are the six smart farming systems presented in Section 1.4.

The complexity and detail of design approaches follow the order: **design mindset** < **architecture** < **platform** < **system**. Consequently, a design approach can be further developed with more detail from the ones on its left side. For example, a new architecture is developed based on the principles of a design mindset.

This research aims to propose a design approach for CASs that focuses on the goal but not the location and type of components of systems. Thus, the design approach should be **architecture**.

## 1.3 Stack of Services for Context-Aware Systems

Stack of services for CASs is an architecture inspired by the microservice design mindset. The microservice design mindset contains the principle to develop a system *"made up of capability-aligned micro-services" (Nadareishvili et al., 2016)*. Each micro-service is *"an independently deployable component of limited scope that supports interoperability through message-based communication. It is small, highly decoupled, and focus on doing a small task" (Newman, 2015)*. Stack of services for CASs is composed of several services organized in order and relies on the context life cycle of CASs. Stack of services for CASs allows system developers to focus on the goal of services. Note that term **service** is a specialization of the term **process** in the situation that the system is highly distributed. The stack of services has the following advantages.

- The system is easy to be upgraded since the goals of services never change when the hardware and software components are replaceable.
- This approach can generate numerous design solutions based on the services available in the system. Then, system designers have more choices to select the one that fits their goal the best.
- The types and numbers of computing devices and software programs grow by time. Fortunately, the number of services is limited. Therefore, it is easier to share knowledge about services than knowledge about computing devices and software components.

The stack of services for CASs has 16 services organized in four phases of the context life cycle of CASs. The 16 services are defined based on the study over several e-agriculture systems and the knowledge of stakeholders in this research project. Figure 7 illustrates the stack of services for CASs. The four squircle blocks, from bottom to top in the vertical order, correspond to the four phases of the context life cycle. Each squircle block contains several services. They are interdependent, also according to the vertical order, from bottom to top. To make it explicit: in a squircle block, when one service X is on top of a service Y means that a new session of the service X starts when the session of the service Y ends. In each squircle block, there exist multiple flows of services. A flow of services represents the possible path of services inside a block, in one direction from bottom to top. A flow starts with the bottom service, moves to the higher service until reaching the top service and terminates the phase. For example, in the acquisition block, the path from the **source selection** service, crossing the **internal data collection** service, towards the **cleaning** service is a flow of services. Another flow of services could be the path from the **source selection** service, crossing the **external data collection** service, towards the **cleaning** service.

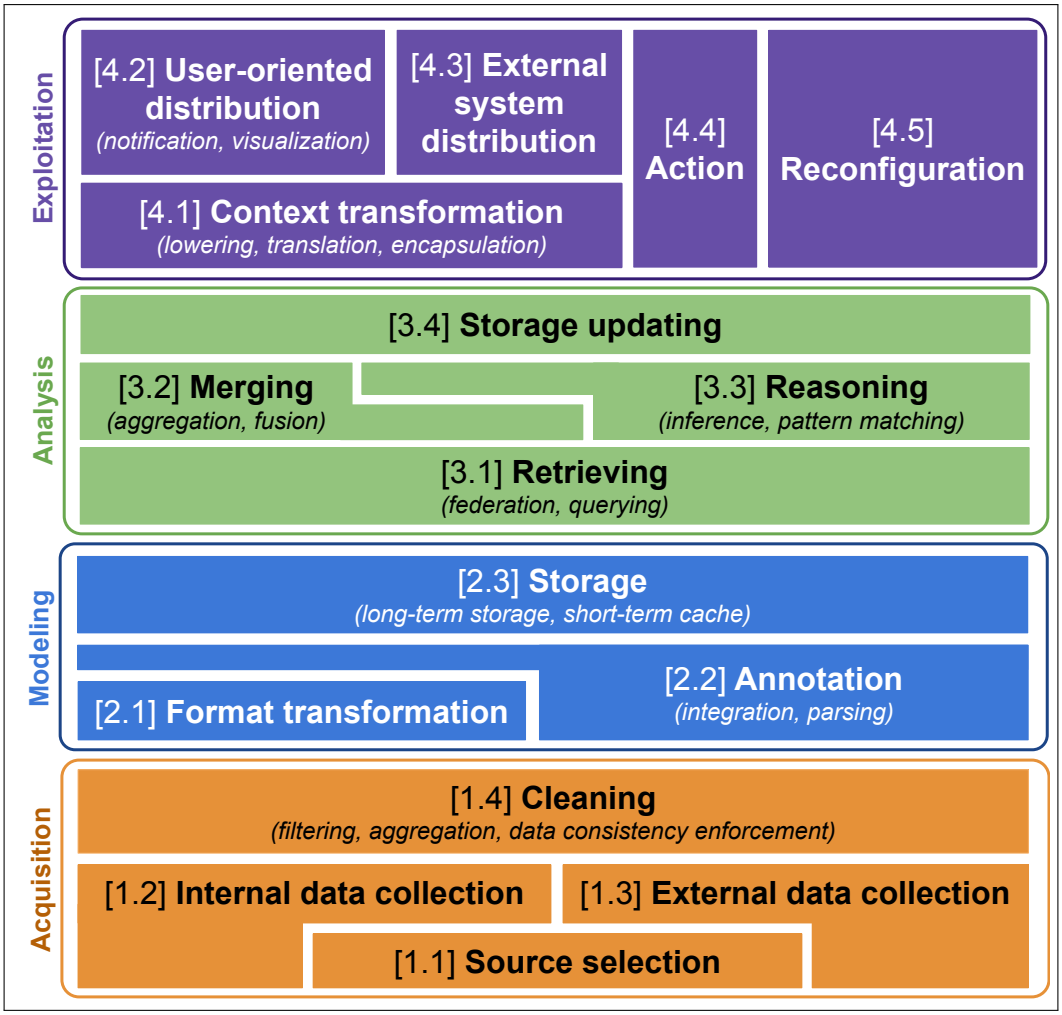



**Figure 7**: Stack of services for CASs

### 1.3.1 Services of the Acquisition Phase

The acquisition phase contains four services as follows.

1.1 **Source selection**: The service to register the configuration data of data sources at run-time. By using configuration data , another service can retrieve data from the data sources. The virtual sensor description of the Global Sensor Network (GSN) middleware is an example of configuration data *(Aberer et al., 2006)*.

1.2 **Internal data collection**: The service to collect data from internal data sources. For example, measured data are collected from a sensor device. This service works differently depending on the communication models: pull model (request/response) and push model (publish/subscribe). In the pull model, the procedure of sending requests and waiting for measured data arrives is repeated.

In the push model, the service subscribes to a source only one time, but it receives new measured data as soon as it is generated.

1.3 **External data collection**: The service to collect data from external data sources. For example, a national weather station that sends forecast data is an external data source. The working procedure of this service also depends on the communication model of pull and push. Different from the internal data collection phase, the external data collection always needs the agreement of the external systems for the data. Moreover, the exchange message between systems must be encapsulated under the networking protocol standards.

1.4 **Cleaning**: The service to transform the input data to guarantee the correctness of data. Three possible actions in the data treatment are **data consistency enforcement**, **filtering**, and **aggregation**. **Data consistency enforcement** replaces error input data with the correct data. It is also named numerical data consistency enforcement *(Sha and Shi, 2008)*. **Filtering** selects relevant data and removes the irrelevant one. **Aggregation** uses basic operators such as addition and subtraction, to produce more accurate data.

### 1.3.2 Services of the Modeling Phase

The modeling phase contains three services as follows.

2.1 **Format transformation**: The service that the format of the data is transformed into another format. For example, a text is extracted from a PDF document and transformed into an XML file.

2.2 **Annotation**: The service to interpret data using a specific schema. The data after this interpretation becomes a part of the low-level context. Two possible actions in the annotation service are **parsing** and **integration**. **Parsing** is the particular case of **integration**: the clean data received from the acquisition phase are sufficient to be represented in the model. For example, a message from a sensor device contains not only the value of measurement but also the relevant metadata such as the unit of measurement, type of sensor device, and the date-time of the measurement. They are enough to be interpreted in the schema. **Integration** is when the clean data needs to be combined with other data stored in the system to be interpreted.

2.3 **Storage**: The service to store data in different storages. Two cases are: to store the data into long term storage or short term cache. Data stored in long term storage is persistent over time. In contrast, short term data is stored in a temporary cache for the short term use. Frequently, the short term cache is for data streaming.

### 1.3.3 Services of the Analysis Phase

The analysis phase contains four services as follows.

3.1 **Retrieving**: The service that retrieves low-level context data stored in the system. This service equals **federation** when the retrieved data are from more than one database. This service equals **querying** when the retrieved data are only from one database.

3.2 **Merging**: The service that uses aggregation operators or fusion to combine several types of low-level context data to produce new low-level context data. This service equals **aggregation** in the case that the input is only numerical data. The **aggregation** operates calculations such as sum, average, min, and max. Otherwise, this service equals **fusion** in the case that the types of input are heterogeneous. An example is the fusion of an imagery map representing farmland and the coordinates of sensor nodes located in the farmland. This fusion results in a new map with multiple points, of which each point corresponds to the position of a sensor device.

3.3 **Reasoning**: The service that uses reasoning techniques to produce new high-level context data. Many reasoning techniques are available in computer science; however, the two techniques that are considered are inference and pattern matching. This service equals **inference** when the system uses a knowledge base to deduct new data. The knowledge base comprises a facts base and a set of rules. The facts base stores low-level and high-level context data of the system. Rules could be interdependent: some rules can only work based on the results derived from firing other rules. An inference made by an inference engine is triggered by the user or by the schedule of an application. This service equals **pattern matching** when the system possesses a set of patterns. Patterns are defined by domain experts or by users. The pattern detection engine observes the input and produces the corresponding output when the input is exactly matching with the pattern. Different from a rule, a pattern is independent of other patterns. One pattern itself contains enough data to produce a decision (high-level context data) from the input.

3.4 **Storage updating**: The service that updates the new data produced after the analysis phase into the storage.

### 1.3.4 Services of the Exploitation Phase

The exploitation phase contains five services as follows.

4.1 **Context transformation**: The service that the high-level context data

received from the analysis phase is transformed into another format. The three possible actions of the context transformations are **lowering**, **translation**, and **encapsulation**. The service equals **lowering** when high-level context data is transformed into low-level context data. The service equals **translation** when data is translated into a human-readable format. The service equals **encapsulation** when data are encapsulated by networking protocol standards to become an encapsulated message.

4.2 **User-oriented distribution**: The service to provide information in a human-readable format to users. An example of the human-readable format could be an imagery file or an HTML web page. Two possible actions of the user-oriented distribution service are **notification** and **visualization**. **Notification** uses the push model: as soon as receiving new data from the context transformation service, a message containing the data is sent to the user. **Visualization** uses the pull model: when the user's device requests for information, a response message is sent to the user.

4.3 **External system distribution**: The service to send encapsulated messages received from the context transformation service to external systems. The encapsulated messages contain data. The working procedure of this service also depends on the communication models of pull and push.

4.4 **Action**: The service to control an actuator device. An actuator can be a simple device such as an open/close water valve.

4.5 **Reconfiguration**: The service to automatically reconfigure the working schedule of the other services according to the changes of context. For example, suppose that the system needs to send a notification to farmers when the soil is dry. The soil is dry, equivalent to the fact that the measurement of the tensiometer generates a data equal or superior to 150 cbar. Given that by default, the tensiometer generates a new measured data and sends it to the server once per day. When the value of the last measurement is 149 cbar, the reconfiguration service detects that the measurement will reach the 150 cbar soon. Then, it should increase the frequency of measurement of the internal data collection service into once per hour.

## 1.4 Six Smart Farming Systems

This section is to review six smart farming systems: `Kirby smart farm system`, `PLANTS system`, `Phenonet-OpenIoT system`, `FarmBeats system`, `BIO-ICT system for precision agriculture`, and `system of Hwang et al`. They are selected from

several systems in e-agriculture. The selection procedure relies on three criteria. First, the selected systems must be CAS, which can automatically change their behaviors according to the context. Second, the selected systems must be published in academics and have real experimentation. Third, the farming activities of smart farming systems relate to crops and livestock.

At first, this research uses the services of the stack of services for CASs to describe each smart farming. The description procedure has three steps. The first step outlines a smart farming system as its original description from the publications related to the system. Note that this research focuses on context and services that handle context, then the hardware devices and software components irrelevant to this goal will be ignored. Moreover, in the original description of some systems, several services exist, but the components handle them are unnamed. Thus, it is necessary to add extra hardware and software components that handle these services. The result of this step is a figure to describe the architecture of the system. The second step searches for possible services corresponding to processes in the smart farming system. The list of possible services is in the stack of services. The third step determines the data exchanges between components inside and outside the system.

Then, this research compares and analyses the six smart farming systems. This work has two folds. First, it proves that all of the possible services of smart farming systems are available in the stack of services for CASs. In other words, the stack of services for CASs can cover all of the smart farming systems. Second, this state of the art points out which services are fundamental, which services are optional.

The six following subsections describe the six smart farming systems. The comparison in detail of them will be presented later in Section 1.5.

### 1.4.1 Kirby Smart Farm

Kirby Farm is a 2800 hectare farmland in Armidale, Australia *(Griffith et al., 2013)*. The main farming activities of this farm are pasture management and livestock monitoring. To perform these activities, the Commonwealth Scientific and Industrial Research Organization (CSIRO) and the University of New England (UNE) develop a smart farming system called Kirby Smart Farm system. The system enables users to visualize real-time monitored data, to define expected agricultural events, to receive notifications and decision supports *(Taylor et al., 2013; Gaire et al., 2013)*. The Kirby Smart Farm system consists of several computing devices. First, a Smart Farm server executes complex computational tasks. Second, collection points and taggle[1] receivers work as gateways. Third, sensor devices, including cattle tags, soil sensor

[1] https://taggle.com.au

devices, and private weather stations, are for the monitoring of different agricultural phenomena. In detail, cattle tags are for livestock's position detecting; soil sensor devices are for soil moisture, temperature, and electrical conductivity measuring; and weather stations are for air temperature measuring.

There are two primary technology contributions in Kirby Smart Farm. The first one is the two improvements for the GSN middleware. GSN is well-known as an open-source middleware for sensor data streaming management *(Aberer et al., 2006)*. The first improvement is to upgrade GSN using semantic web technologies: transforming raw data into RDF data. The RDF data can be published on the web following linked open data (LOD) standards[2]. Kirby Smart Farm uses an ontology combined of the semantic sensor network (SSN) ontology *(Compton et al., 2012b)* and agricultural domain-specific ontologies, to model the data. The second improvement to GSN is that Kirby Smart Farm allows users to contribute their knowledge by defining self alerts and events using the complex event processing (CEP) framework *(Taylor and Leidinger, 2011)*. In addition to the upgrade of GSN, the second technology contribution of Kirby Smart Farm is the combination of multiple technologies such as GPS, satellite imagery and sensory data, in providing a spatially-enabled planning and management dataset for multiple objectives.

Figure 8 describes the components, services, and flows of data, of the Kirby Smart Farm system. To ease the description of this system, five extra components are added: *stream processing*, *storage*, *external source*, *user's device*, *RDF data distributor*, and *external system*. First, the *stream processing* component handles the data annotation, data organizing and data querying. Second, the *storage* component stores virtual sensor description files and dataset. Third, the *external source* represents the sources of external data. Fourth, the *user's device* component represents devices of users. Fifth, the *RDF data distributor* handles distributing the RDF data to other external systems. Last, the *external system* represents the other systems that access the data of the Kirby Smart Farm system.

In the acquisition phase, at first, it is necessary to determine internal data sources and external data sources. Internal data are measured data retrieved from *soil sensor devices*, *weather stations*, and *cattle tags* in the local farmland. External data are events defined by users through the *web display portal*. The **source selection** service is run by the *virtual sensor manager*. This service is to select the internal data sources recorded in virtual sensor description files. These files are stored at the *storage*. Then, the **internal data collection** service are run by the *collection points* and the *taggle receivers*. The *collection points* collect measured data from *weather stations* and *soil sensors*. The *taggle receivers* collect measured data from *cattle tags*. These measured

[2] https://www.w3.org/DesignIssues/LinkedData.html

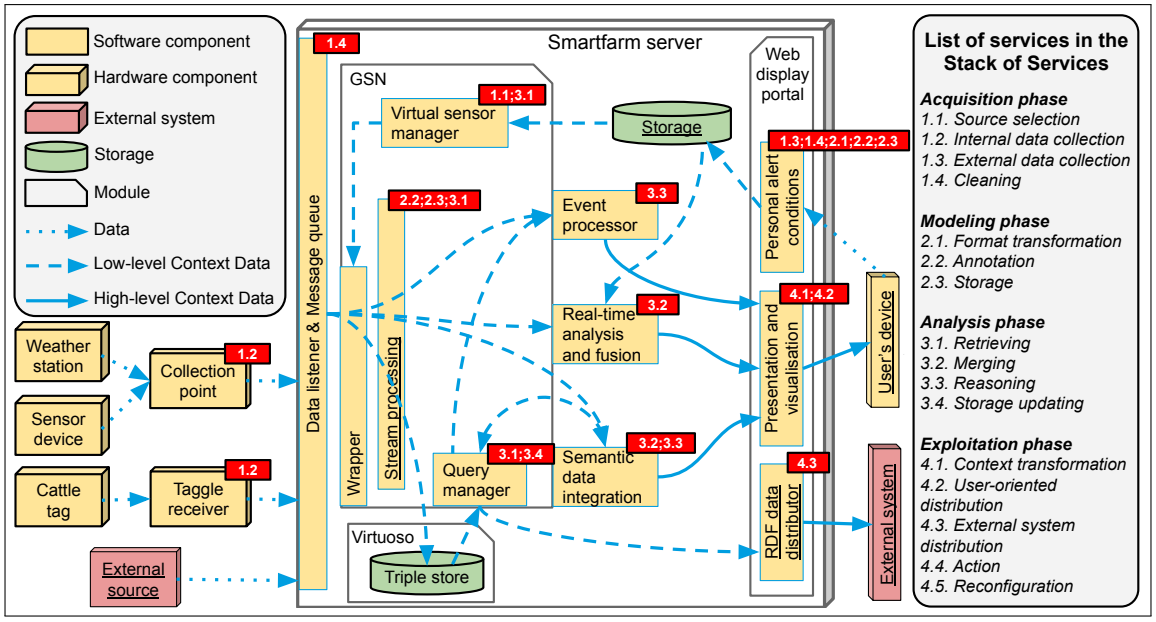



**Figure 8**: Description of the Kirby Smart Farm system

data are transferred to *Smart Farm server* where they are filtered, corrected, and put in corresponding queues due to the **cleaning** service run by the *data listener & message queue* component. About external data, the **external data collection** service enables users to define events and alerts via the *personal alert conditions* component in the *web display portal*. This component runs the **cleaning** service to guarantee that input data from humans are correct. In addition, the *wrapper* component of the *GSN* also supports to collect data from the external sources. Thus, the *wrapper* component could run the **external data collection** service.

The modeling phase is to integrate internal data streams and external data into the system. The *data listener & message queue* component leads internal data streams to the *wrapper* component of the GSN module. Based on the description of virtual sensors, this component forwards input streams to appropriate outputs. Then, the *stream processor* component runs the **annotation** and **storage** services to handle the data streams. The **annotation** service annotates data streams with the information of virtual sensors and the related data from the ontology to transform them into RDF data streams. The **storage** service is to organize RDF data in the *Virtuoso triple store*. Meanwhile, the external source data collected in the *web display portal* are translated to event description files due to the **format transformation** service before being saved in the *SF storage* due to the **storage** service.

In the analysis phase, the data streams retrieved by the *stream processing* component due to the **retrieving** service are forwarded to the following components:

*real-time analysis and fusion*, *semantic data integration* and *event processor*. First, the *real-time analysis and function* component runs the **merging** service to combine the data streams with the other available data set in *SF storage* such as GPS-based asset mapping and satellite imagery to produce a spatially-enabled planning and management data set. This data set is further used as a source for further analysis. Second, the *semantic data integration* component runs **merging** service to do calculations to produce low-level context data and runs **reasoning** service to produce high-level context data according to users' demands. Third, the *event processor* component runs the **reasoning** service to produce high-level context data in the form of alerts when the input streams are matched with users' defined events. All data and context produced by the previous services are stored in the *triple store* or the *SF storage* due to the **storage updating** service.

In the exploitation phase, the *presentation and visualisation* component runs the **context transformation** service to translate context and data into a human-readable format. Also, this component runs the **user-oriented distribution** service to send the human-readable context and data to users. Moreover, the RDF data stored in the *virtuoso* can be queried by the *external system* due to the **external system distribution** service. This service is run by *RDF data distributor* component.

### 1.4.2 PLANTS

The smart farming system developed in the PLANTS project (PLANTS system) refers to as an autonomous irrigation system *(Goumopoulos et al., 2009)*. The system performs farming activities, including plant growth real-time monitoring and irrigation control. The experiment of this system is in a greenhouse at the University College Cork, Ireland *(Goumopoulos et al., 2014)*. The greenhouse embraces the farmland of 96 strawberries equipped with PAM[3] meters, sensor and actuator devices. Sensor devices measure the temperature of the leaves of plants, ambient temperature and soil moisture. Actuator device is an irrigation distribution unit that waters the crops. The other hardware components are gateways, driver operators, a coordinator node and a web server *(Goumopoulos, 2012)*. The coordinator node is a server with high computational capability. A driver operator is defined as a hardware component installed with a specific driver that connects to a gateway of each wireless sensor and actuator network (WSAN).

The PLANTS system has two advantages. First, a knowledge base containing a rule base and a fact base enables the system to infer new data and make decisions. A

[3]Pulse-Amplitude-Modulation chlorophyll fluorometer is a device used for chlorophyll fluorescence analysis.

specific ontology dedicated to this system is used to model data. The knowledge base is also a part of the decision support system (DSS) module in the PLANTS system. Second, this system uses machine learning and expert knowledge in enriching the rule base. It uses log data of the system, combined with support from data mining experts and agriculture experts, to produce the new appropriate rules, and put them in the rule base. However, the second advantage is out of the scope of this review because it has no impact on the flows of data.

Figure 9 describes the components, services, and flows of data, of the PLANTS system. To ease the description of this system, four extra components are added: *local device interface*, *fact base*, *external source* and *user's device*. First, the *local device interface* component is the interface between the *coordinate node* and the *driver operator*. Second, the *fact base* storage contains data modelled using the PLANTS ontology. Third, the *external source* represents sources of external data such as a weather station. Finally, the *user's device* represents devices of users. On the other hand, some components in the PLANTS system are removed from this analysis such as *interaction manager*, *communication manager* and *machine learning* since they have no impact on flows of data.

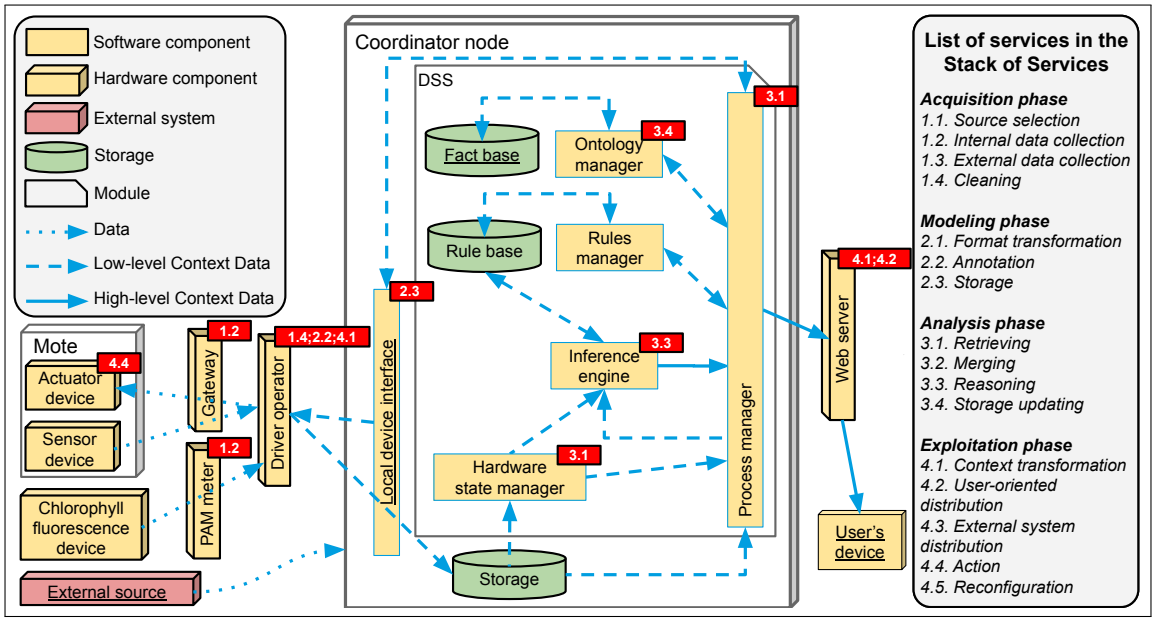



**Figure 9**: Description of the PLANTS system

In the acquisition phase, the PLANTS system collects measured data from sensor devices. The *gateway* runs the **internal data collection** service to collect measured data from the *sensor nodes* in the *motes*. Also, the *PAM meter* also runs the **internal data collection** service to collect measured data from the *chlorophyll fluorescence devices*. The measured data are transferred to the *driver operator*. This component

runs the **cleaning** service that corrects and filters the data. Moreover, it is worth to note that the PLANTS system could provide the **external data collection** service to collect external data from *external source*. In a publication of the PLANTS system, the authors note that the *external source* could be a weather station; however, there is no further detail about this service.

In the modeling phase, the *driver operator* runs the *annotation* service to combine extra data such as timestamps with the data derived from the *gateways* and *PAM meters*. The data contains both measured data and the state of sensor devices. These data are represented and could be understood by the coordinated-node; then, they are the low-level context data. The *local device interface* component in the *coordinator node* receives these data and runs the **storage** service to organize them into the appropriate places: data stored in the database and the state of sensors stored in devices' state repository in the *storage*.

In the analysis phase, the *process manager* component is responsible to connect all software components in the *coordinator node*. The *process manager* component runs the **retrieving** service to retrieve data from the *storage* and distributes them to appropriate components. It can retrieve data in RDF form from the *fact base* through the *ontology manager* component. Then, it transfers these data to *inference engine*. The hardware state manager component retrieves the state of devices and distributes them to the *inference engine*. The *inference engine* component runs the **reasoning** service to produce high-level context data. All the new contexts are updated into the *fact base* due to the **storage updating** service run by the *ontology manager* component.

In the exploitation phase, the system operates two activities: controlling the irrigation actuator devices and sending information to users. In the first activity, the *actuator device* runs the **action** service to spray mist in the greenhouse or water some plants. In the second activity, the context and data are converted into human-readable form due to the **context transformation** service. Then, these human-readable form data are sent to the *user's device* due to the **user-oriented distribution** service. These two services are run by the *web server*.

### 1.4.3 Phenonet-OpenIoT

Phenonet-OpenIoT is a smart farming system developed by CSIRO *(Jayaraman et al., 2015)*. This smart farming system is based on the OpenIoT platform *(Soldatos et al., 2015)*. The experiment of the Phenonet-OpenIoT system is in a study plot named Kirkegaard and Danish, in Australia *(Jayaraman et al., 2016)*. Domesticated species in this experiment are crops used as food for sheep. However, there is no information about the type of crops in the related publication. The farming activities are, at first,

to monitor soil moisture, water use, crop growth rate and crop yield, then to evaluate the effect of sheep grazing on crop re-growth. The sensor devices used for monitoring works are soil moisture at multiple depths and canopy temperature sensor devices. The next two elements of the Phenonet-OpenIoT system are a gateway and a cloud server. The gateway uses the X-GSN (extension of GSN) middleware *(Calbimonte et al., 2014)*; then, it can semantically annotate data using the vocabularies of the ontologies such as SSN, GEO[4] and QU[5]. The cloud server contains some modules of the OpenIoT platform such as the Linked Stream Middleware (LSM), the scheduler and the Service Delivery and Utility Manager (SDUM). An LSM is a module that handles cloud data storage. A scheduler is represented as a software component that processes users requests and ensures access to the resources. An SDUM is represented as a software component that registers users with their requests.

The Phenonet-OpenIoT system has two advantages. First, a Do-it-Yourself (DIY) application that allows users to define the interface of virtual devices and expectable results. Second, this system develops the Phenonet ontology, which is the combination of the SSN ontology and some extension for agriculture. The goal of this ontology is to improve the interoperability of the system.

Figure 10 describes the components, services, and flows of data, of the Phenonet-OpenIoT system. To ease the description of this system, five extra components are added: *virtual sensor configure base*, *LSM directory storage*, *LSM data access*, *user's device* and *external source*. First, the *virtual sensor configure base* contains virtual sensor descriptions at the local side. Second, the *LSM directory storage* also stores virtual sensor descriptions but in the cloud server. Third, the *LSM data access* component is an interface for the *LSM* module that interacts with the other components in the *cloud server*. Fourth, the *user's device* represents devices of users. Finally, the *external source* represents the source of external data.

In the acquisition phase, the gateway collects measured data from the *sensor nodes*. At first, the *virtual sensor manager* component runs the **source selection** service to select data streams based on the virtual sensor descriptions stored in the *virtual sensor config base*. Next, the *wrapper* component runs the **internal data collection** service to retrieve measured data from internal sources. The *wrapper* component of the *GSN* also supports collecting data from the external sources. Thus, the *wrapper* component could also run the **external data collection** service. Moreover, this component runs the **cleaning** service to filter and aggregate the collected data streams using time-based windows.

In the modeling phase, the *annotation* component runs the **annotation** service to

[4] https://www.w3.org/2005/Incubator/geo/

[5] https://www.w3.org/2005/Incubator/ssn/ssnx/qu/qu

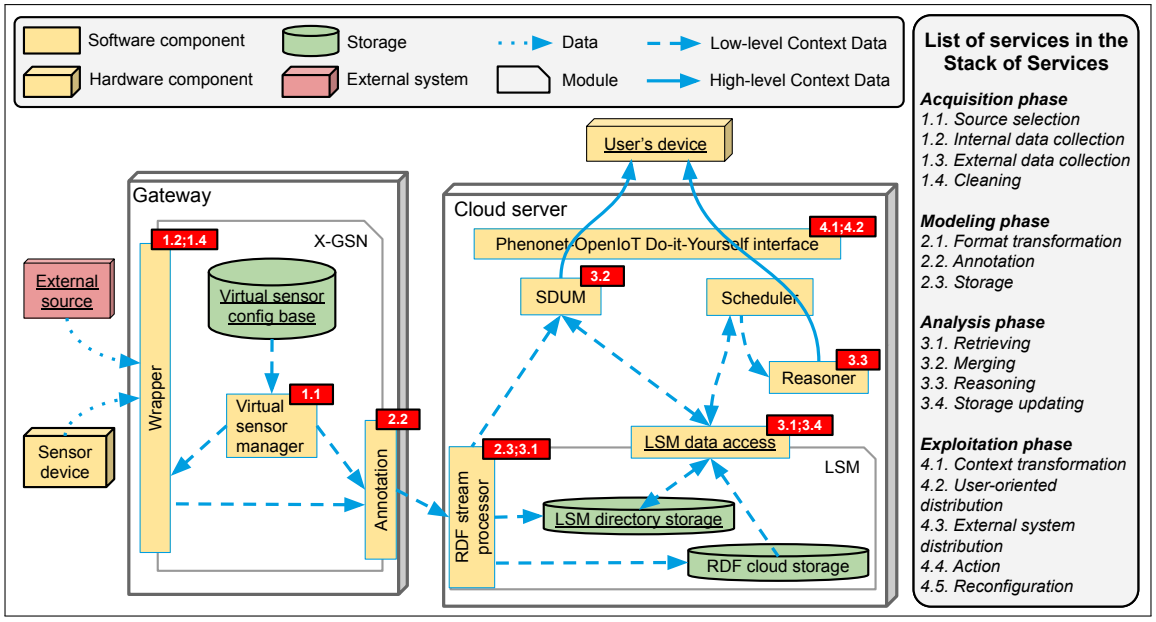



**Figure 10**: Description of the Phenonet-OpenIoT system

annotate the data streams using the vocabulary of the Phenonet-OpenIoT ontology. Then, in the cloud server, the *RDF stream processor* runs the **storage** service to organize the data into the *RDF cloud storage*. After being annotated and represented, the data are low-level context data. Moreover, the virtual sensor descriptions in the gateway are transferred to the cloud server. These descriptions are stored in the *LSM directory storage* due to the **storage** service.

In the analysis phase, the low-level context data are processed differently depending on activities: sensor discovery or data retrieving. In the case of sensor discovery, at first, *LSM data access* component runs the **retrieving** service to get the list of virtual sensors. When users request the target sensor devices and observed phenomenon, the *reasoner* component runs the **reasoning** service to produce a filtered list of virtual sensors appropriate to the users' request. In the case of data retrieving applications, the system provides continuous or static data depending on the demand of users. On the one hand, the *RDF stream processor* component runs the **retrieving** service to get real-time data streams. On the other hand, the *LSM data access* component runs the **retrieving** service to get static data from the *RDF cloud storage*. The *SDUM* component uses the SPARQL scripts retrieved from *LSM directory storage* to query the data. The querying activities also include aggregation. Thus, it is possible to conclude that the *LSM directory storage* also runs the **merging** service. The results of the **reasoning** service are stored in the cloud server due to the **storage updating** service.

In the exploitation phase, the **context transformation** service translates high-level context data into human-readable form. Then, the **user-oriented distribution** service sends this content to *users' devices*. The two services are run by the *Phenonet-OpenIoT Do-it-Yourself interface* component.

### 1.4.4 FarmBeats

FarmBeats is a smart farming system developed by Microsoft *(Vasisht et al., 2017)*. The system develops applications for the precision agriculture domain such as irrigation decision, crop harvesting suggestion and livestock management. FarmBeats experienced six months in two states of the United States of American: one in upstate New York and one in Washington DC. The experimentation in New York is a 100acres (about 40 hectares) farmland planted with vegetables, fruit, grains, combined with raising livestock for dairy and meat products. In the experimentation in Washington DC, farmers grow vegetables on farmland with an area of five acres (about two hectares). The system contains an Azure Cloud, a FarmBeats gateway, IoT base stations and multiple sensor devices. The FarmBeats gateway could be considered as a local server based on its computational and storage capabilities. An IoT base station handles communication between sensor devices and the FarmBeats gateway. Most of the sensor devices measure the pH, moisture, temperature of the soil. Some sensor devices generate imagery of the farmland. Actuator device is called an irrigation unit, and it is responsible for watering the crops.

The first advantage of FarmBeats is that it uses various types of sensor devices to improve the consistency of the input data. They include ten types of sensors, three types of cameras and three types of drones. Moreover, the combination of different types of data can produce more informative data. For instance, farmland images from drones associated with the location of sensor devices and measurement values from the WSN provide a precision map that supports the precision farming activities. This new map contains more useful information for users than the original one. The second advantage of this system is to use machine learning in prediction of the situation of the entire farm based on the situation of some discrete locations. The third advantage of this project is in video and imagery treatment: it combines sparse 3D reconstruction techniques from video with image stitching techniques.

Figure 11 describes the components, services, and flows of data, of the FarmBeats system. To ease the description of this system, five extra components are added: *server computing*, *cache*, *local users' device*, *global user's device* and *other farming system*. First, the *server computing* component handles calculation and storage tasks in the *Azure cloud server*. Second, the *cache* is the cache to store temporary data at the FarmBeats gateway. Next, the *local users' device* and *global user's device*

represent devices of users respectively at the local network and on the Internet. Finally, the *other farming system* represents the other farming systems that share farming analytics with the FarmBeats system.

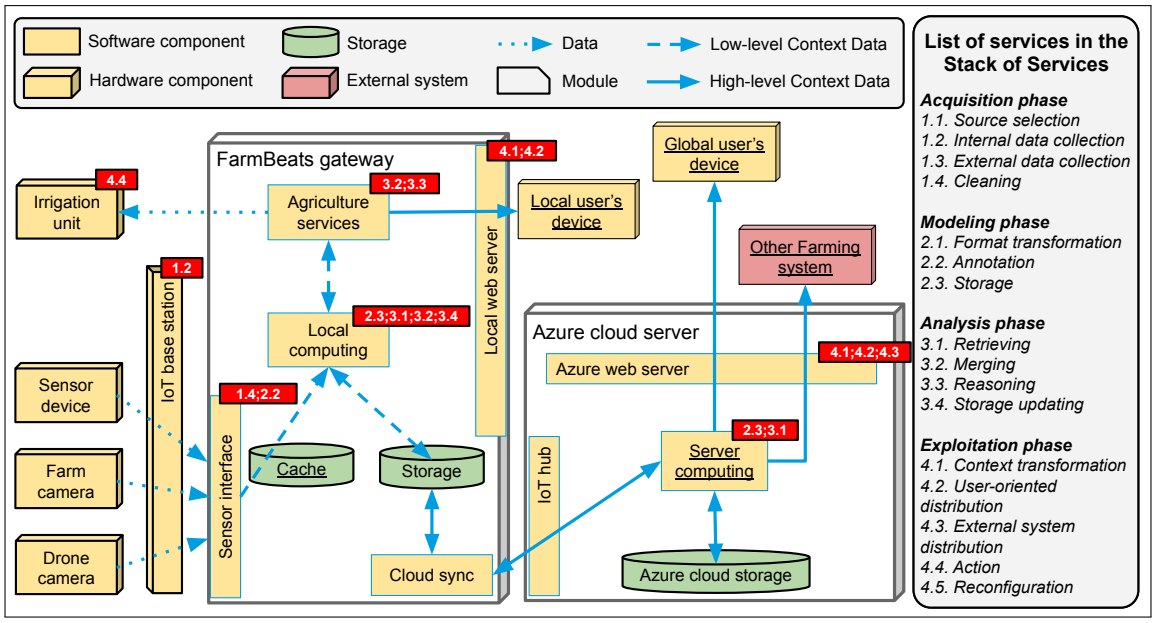



**Figure 11**: Description of the FarmBeats system

In the acquisition phase, the *IoT base station* runs the **internal data collection** service to collect measured data from the *sensor devices* and to collect images from the *farm cameras* and *drone cameras*. These data are transferred to the *FarmBeats gateway*. The *sensor interface* component in the *FarmBeats gateway* runs the **cleaning** service to guarantee that the data can be further used. For instance, the mechanism of the MQTT, the network protocol used for the communication between the *IoT base station* and the *FarmBeats gateway*, is to ignore inappropriate data.

In the modeling phase, the *sensor interface* component runs the **annotation** service to add related data to the received data and to represent them at the *FarmBeats gateway*. The **storage** service is to organize all the data in the appropriate storages. This service is run at the both *FarmBeats gateway* and *Azure cloud server*. In the former case, the long-term data are stored in the *storage* component, and the short-term data are stored in the *cache*. Note that the short-term data are for real-time data visualisation. In the latter case, the data, at first, are transferred to the *Azure cloud server*. Note that only the high-level context data inferred after the analysis phase, are transferred to the *Azure cloud server* for long-term uses. Then, the *server computing* component organizes the data into the *Azure cloud storage*.

In the analysis phase, at the *FarmBeats gateway*, the *local computing* component

runs **retrieving** service to retrieve data from local *storage*. Next, the *local computing* component runs the **merging** service to produce a precision map integrated with aggregate data such as daily average measured data. This is the result of both the fusion between image and numerical data and the aggregation of numerical data. Moreover, this component runs the **reasoning** service to produce high-level context data such as crop suggestions and livestock health statistics. All of the new data are stored in the *storage* due to the **store updating** service. At the *Azure cloud server*, all the data stored in the *Azure cloud storage* can be retrieved by the **retrieving** service run by the *server computing* component.

In the exploitation phase, the *web server* at the local side runs the **context transformation** service and **user-oriented distribution** service to send the human-readable data to the *local user's devices*. In the *Azure cloud server*, the *Azure web server* runs the same services to send human-readable data to the *global user's devices*. Moreover, at the local side, the *irrigation units* run the **action** service to water the crops. It is worth to mention that this smart farming system also provides the **external system distribution** service since it supports cross-farms analytics; however; there is no information about this service in the system's publication. Suppose that the *Azure web server* runs the **external system distribution** service to send data to the *other farming system*.

### 1.4.5 BIO-ICT System for Precision Irrigation

BIO-ICT is a project of BIO-ICT Centre of Excellence in Montenegro *(Bajceta et al., 2016)*. The project aims to build a general IoT architecture for three different use cases: precision agriculture, aquaculture monitoring and environment monitoring *(Popovic et al., 2017)*. Each use case addresses a different group of stakeholders. This review focuses only on the use case of precision agriculture which has two missions: soil protection and plant protection. The soil protection mission is about doing analytics in irrigation and fertilization. The latter mission is for the grapevine diseases treatment and testing the resistance of plants. The BIO-ICT system for precision irrigation consists of a WSN with multiple types of sensor devices, actuator devices, gateways and a private cloud server. This system experimented in a vineyard in Danilovgrad, Montenegro. Sensor devices measure air temperature, relative humidity, pressure, soil temperature, wind speed and leaf wetness. The actuator device is a smart spraying unit to water the crops.

The BIO-ICT system focuses on providing API between the private cloud server, and users' devices or third-parties systems. Moreover, it provides flexible tools for different stakeholders: the data collection tool addresses data scientists, and the plant disease detection tool addresses farmers.

Figure 12 describes the components, services, and flows of data, of the BIO-ICT system for smart spraying. To ease the description of this system, three extra components are added: *cache*, *users' device* and *third-party application system*. The *cache* component represents a table in the database at the *cloud server* for the short-term data. The *users' device* represents devices of users. The *third-party application system* represents external systems that can access the database of the BIO-ICT system and provide application from such data. Inversely, the *spraying unit* is removed from this analysis since users manually control it through the server.

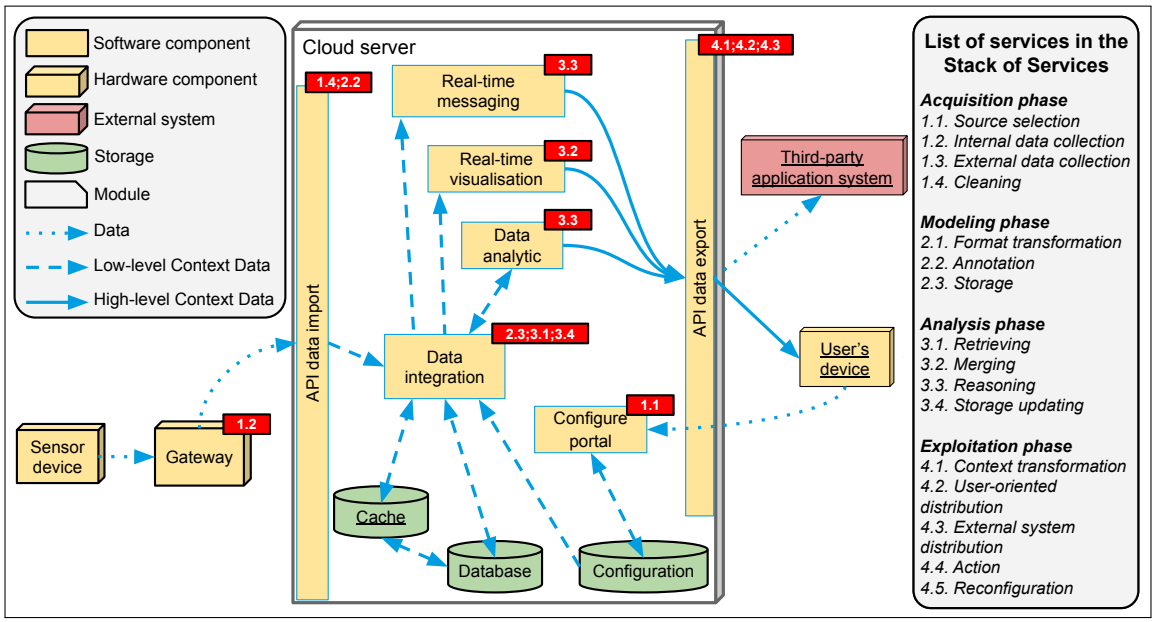



**Figure 12**: Description of the precision irrigation use case of the BIO-ICT system

In the acquisition phase, the *configure portal* component runs the **source selection** service to access the virtual sensor devices stored in *configuration* storage. A virtual sensor device is defined by users in the form of a description file. The description file includes the information of name, location and type measurement of the virtual sensor device. The *gateway* runs the **internal data collection** service to collect measured data from sensor devices. Then, the measured data are transferred to the *cloud server*. Next, the *API data import* component runs the **cleaning** service to ensures that the data can be used further. For instance, the data in the messages with the inappropriate API keys could be ignored.

In the modeling phase, the *API data format* component runs the **annotation** service to represent the data. Then, the data is in system-understandable format and becomes low-level context data. Next, the *data integration* component runs the **storage** service to organize data in the *database* or in the *cache*. On the one hand, the *database* stores all the data. On the other hand, the *cache* only stores the 20

lasts measured data of each sensor device, and these data are used for real-time data visualization.

In the analysis phase, the *data integration* component runs the **retrieving** service to retrieve data from the *database* and the *cache*. These data are transferred to appropriate components. These components are: *real-time messaging*, *real-time visualization* and *data analytic*. The *real-time visualisation* component runs the **merging** service to provide informative data from the measured data and its related data stored in the *cache*. The **reasoning** service run by the *real-time messaging* and *data analytic* components produces high-level context data for the human alerting and the disease forecasting.

In the exploitation phase, the *API data export* component runs the **context transformation** and **user-oriented distribution** services to transform context in human-readable form and send them to users' devices. The **context transformation** service can convert the data into CSV or JSON file formats. Moreover, the *API data export* also provides the data to the *third-party application system*.

### 1.4.6 System of Hwang et al.

The system developed by Hwang et al. aims to manage the growth of paprika in a greenhouse in Jeollanam-do, Korean *(Hwang and Yoe, 2011; Hwang et al., 2010)*. The system supports to improve the production and yield of paprika by precision irrigation and heating technologies. In the greenhouse, various sensor devices are installed. First, the sensor devices dedicated to the ambience, measure temperature, humidity, illumination and CO2 levels. Second, the sensor devices dedicated to the crops, measure leaf humidity and temperature, body weight and height, and fruit temperature and volume. Third, the sensor devices dedicated to the soil, measure moisture, temperature and pH. Also, the greenhouse is equipped with several actuator devices for activities such as illuminating, watering, CO2 generating, heating and ventilation. The actuator devices are called greenhouse facitlities. They are both manually controlled by users or automatically controlled by the system. One the one hand, the sensor devices connect to the middleware through the gateways. On the other hand, each actuator device connects to the middleware through Program Logic Controller devices (PLC). The middleware connects to a Paprika Greenhouse Management Server which includes a database server, an application server and a web server. Furthermore, this system is equipped with camera systems (CCTV) and image processors to process imagery data.

The system of Hwang et al. focuses on developing a middleware that can collect data from various types of sensor and can control several types of actuator devices. Moreover, the system provides two applications corresponding to two different types

of reasoning. First, the event management application with the pattern detection technique is for real-time notifications. Second, the context-aware management application using the inference technique aims to control the greenhouse facilities. The context-aware management application possesses a self-developed ontology to model networks, sensors, context and activities of the system.

Figure 13 describes the components, services, and flows of data, of the system of Hwang et al. To ease the description of this system, two extra components are added: *server cache* and *users' device*. First, the *server cache* component is the cache to store temporary data at the parika greenhouse management server. Second, the *user's device* represents devices of users. Moreover, the external CCTVs of the system are ignored in this paper because they have no impact on the development of crops.

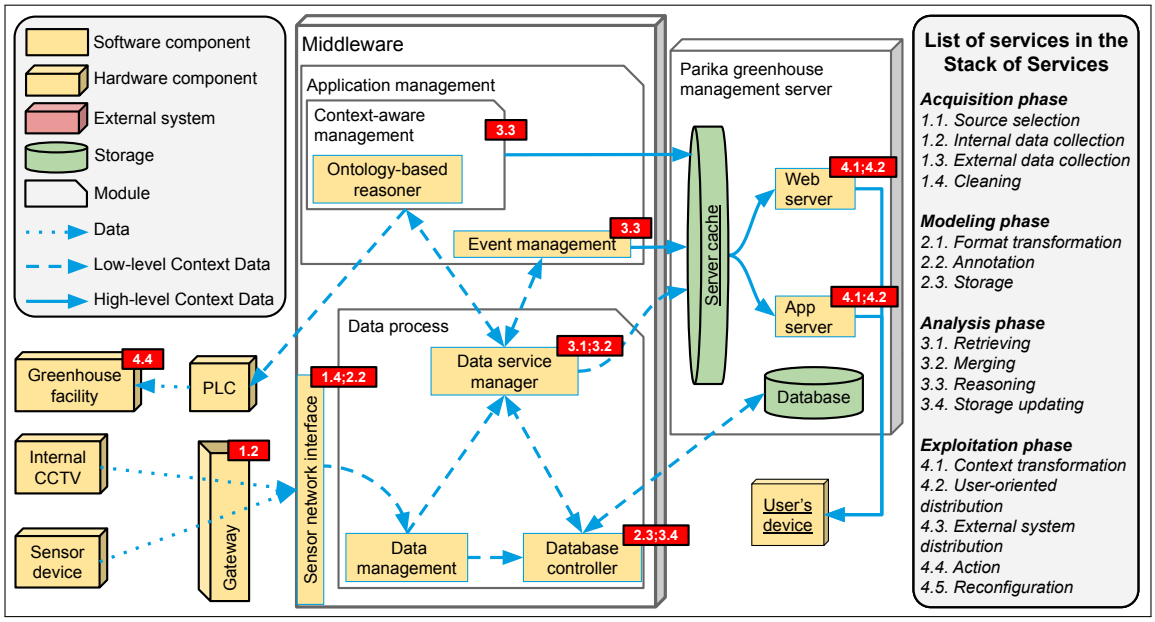



**Figure 13**: Description of the system of Hwang et al.

In the acquisition phase, the *gateway* runs the **internal data collection** service to collect measured data from the *sensor devices* and images from the *internal CCTVs*. The data are transferred to the *sensor network interface* component. This component runs the **cleaning** service to guarantee that the other software components could use the received data. For example, this service converts different data structure types into integer data.

In the modeling phase, the *data management* component runs the **annotation** service to annotate data and represented them. In the case of *context-aware management* application, data are annotated and represent using the self-developed ontology. The *database controller* component runs the **storage** service to store data

in the *database*. Note that this service has a filtering mechanism that only organizes the data when they are new. The *database* is located in the *paprika greenhouse management server*.

In the analysis phase, the *data service manager* component runs the **retrieving** service. This service queries real-time data from the *data management* component or static data from the *database* through the *database controller* component. The queries, themselves, do aggregation. Thus, the *data service manager* component also runs the **merging** service. The real-time data are forwarded to the *parika greenhouse management server* and to the *event management* component. The static data are forwarded to the *context-aware management* component and to the *event management* component. The *event management* component runs the **reasoning** service to produce high-level context data. In detail, the service uses the *pattern matching* technique in this case. Of which, an event is a pattern of data. When the arrived data matches an event, the *event management* component produces the corresponding informative data that are relevant to the alert activity. The *context-aware management* component also runs the **reasoning** service to produce the informative data related to the control of the actuator devices. The service, in this case, applies the *inference* technique by using an *ontology-based reasoner* to infer high-level context data. The *database controller* component runs the **storage updating** service to organize new data from the **reasoning** service into the *database*.

In the exploitation phase, there are two main activities: actuator devices controlling and user content providing. In the first activity, the *greenhouse facilities* such as watering and heating systems, run the **action** service to control the environment in the greenhouse. In the second activities, the *web server* and the *app server* translate data and context retrieved from the *middleware* into human-readable form due to the **context transformation** service. Then, they distribute them to *users' devices* due to the **user-oriented distribution** service.

## 1.5 Comparison of Six Smart Farming Systems

This section analyses and compares the six smart farming systems presented in the previous section. Despite the differences between the six systems in hardware components, software components, and communication technologies, the stack of services enables viewing them under a unified vision. Table 1 shows the list of services by each smart farming system.

The first row of the table includes the six projects corresponding to the six smart farming systems. The first column lists the services in the stack of services. The intersection of a column and a row is called a box. A box checked by the "x"

| | | Kirby Smart Farm | PLANTS | Phenonet-OpenIoT | FarmBeats | BIO-ICT | Hwang et al. |
|---|---|---|---|---|---|---|---|
| **Acquisition** | 1.1. Source selection | x | | x | | x | |
| | 1.2. Internal data collection | x | x | x | x | x | x |
| | 1.3. External data collection | x | x | x | | | |
| | 1.4. Cleaning | x | x | x | x | x | x |
| **Modeling** | 2.1. Format transformation | x | | | | | |
| | 2.2. Annotation | x | x | x | x | x | x |
| | 2.3. Storage | x | x | x | x | x | x |
| **Analysis** | 3.1. Retrieving | x | x | x | x | x | x |
| | 3.2. Merging | x | | x | x | x | x |
| | 3.3. Reasoning | x | x | x | x | x | x |
| | 3.4. Storage updating | x | x | x | x | x | x |
| **Exploitation** | 4.1. Context transformation | x | x | x | x | x | x |
| | 4.2. User-oriented distribution | x | x | x | x | x | x |
| | 4.3. External system distribution | x | | | x | x | |
| | 4.4. Action | | x | | x | | x |
| | 4.6. Reconfiguration | | | | | | |

**Table 1**: Services found in the six smart farming systems

symbol means that the smart farming system, at the first row on the same column with the box, provides the service at the first row on the same column with the box. From this table, there are some following remarks.

- It is possible to conclude that the stack of services for CASs can cover all the smart farming systems.
- Eight services are fundamental: **internal data collection**, **cleaning**, **annotation**, **storage**, **retrieving**, **reasoning**, **storage updating**, **context transformation**, and **user-oriented distribution**. Therefore, the irrigation CAS developed for TSCF should include them.

- Eight services are optional: **source selection**, **external data collection**, **format transformation**, **merging**, **external system distribution**, **action**, and **reconfiguration**. In other words, a smart farming system can work properly without them. Moreover, no system among the six smart farming systems provides the **reconfiguration** service. This service has a side meaning: a system with this service is adaptive since it can modify its working habit to satisfy its context. Therefore, none of the above systems is adaptive.

## 1.6 Summary and Discussion

To sum up, this chapter presents the stack of services for CASs. It is an architecture that relies on the context life cycle of CASs. This architecture follows the principles of the microservice design mindset: it consists of 16 services. The 16 services are identified based on numerous CASs in e-agriculture and the knowledge of computer scientists involved in this project. The stack of services for CASs can improve the upgradability since each service focuses on goal; then, system developers can replace hardware and software components with new ones as long as they maintain the service's goal. Moreover, this chapter reviews six smart farming systems based on the services of the stack of services for CASs and to conclude that its services can cover all of the smart farming systems.

It is possible to conclude that this chapter success in responding to the challenge of the upgradability of CASs. However, there are still two points to discuss further:

- Beside e-agriculture, the stack of services for CASs should be evaluated in other domains, such as smart city and smart transportation, to guarantee that the list of services in the stack is complete. It is necessary to publish and promote this theory so that it gets more contributions and suggestions from the other communities.
- As a side point in Section 1.5, there is no work on adaptive context-aware systems (ACAS). ACAS is a CAS which can *"modify its behavior according to changes in the application's context" (Efstratiou, 2004)*. It could be a perspective for new researches in the future.

# Chapter 2

# Context-Aware System Ontology for Data Heterogeneity

*"The alchemist picked up a book that someone in the caravan had brought. Leafing through the pages, he found a story about Narcissus. The alchemist knew the legend of Narcissus, a youth who knelt daily beside a lake to contemplate his own beauty. He was so fascinated by himself that, one morning, he fell into the lake and drowned. At the spot where he fell, a flower was born, which was called the narcissus. But this was not how the author of the book ended the story. He said that when Narcissus died, the goddesses of the forest appeared and found the lake, which had been fresh water, transformed into a lake of salty tears. "Why do you weep?" the goddesses asked. "I weep for Narcissus" the lake replied. "Ah, it is no surprise that you weep for Narcissus," they said, "for though we always pursued him in the forest, you alone could contemplate his beauty close at hand." "But... was Narcissus beautiful?" the lake asked. "Who better than you to know that?" the goddesses asked in wonder. "After all, it was by your banks that he knelt each day to contemplate himself!" The lake was silent for some time. Finally, it said: "I weep for Narcissus, but I never noticed that Narcissus was beautiful. I weep because, each time he knelt beside my banks, I could see, in the depths of his eyes, my own beauty reflected." "What a lovely story," the alchemist thought."* – **Paulo Coelho**

---

An advantage of using the stack of services for CASs to design a system is to determine the type of contexts (low-level or high-level). Contexts have a meaning in representing a CAS; however, determining data is more significant in processing. Technically speaking, the low-level context and the high-level context are two sets of data. Therefore, representing different types of data and different types of computations that process data are essential.

Even that data in IoT are heterogeneous and numerous, it is possible groups them into three categories: measured data, aggregate data, and deducted data. In CAS, deducted data usually is the state of an entity and is a part of the high-level context. The other ones usually are numeric. They are parts of the low-level context. The computations that produce measured data, aggregate data, and deducted data, respectively are measurement, aggregation, and deduction. This chapter aims to present a new ontology that models these three type of data and their corresponding computations. Consequently, it improves the interoperability of CASs in the IoT. Moreover, this ontology contains the vocabulary that embodies a mechanism to create generic rules for reasoning.

This chapter is organized as follows. Section 2.1 gives more information about ontology and its role in the stack of interoperability. Section 2.2 presents two well-known ontologies SOSA/SSN and SAREF, before selecting one of them as the core to develop a new ontology for CASs. This chapter's main contribution is Section 2.3: to present CASO as the ontology indicated to CASs. Next, Section 2.4 explains the method to evaluate and maintain CASO. Finally, Section 2.5 sums up the chapter and open a discussion.

## 2.1 Ontological Approach for the Interoperability of Data Semantic

The Association Francophone des Utilisateurs de Logiciels Libres (AFUL) defined the interoperability[1] as *"a characteristic of a product or system, whose interfaces are completely understood, to work with other products or systems, present or future, in either implementation or access, without any restrictions"*. There exist several stacks of interoperability. One one hand, most of them consider four interoperability layers of technical, syntactic, semantic, and organization *(Rahman and Hussain, 2020; Kubicek et al., 2011)*. On the other hand, other standards, such as the interoperability convention document for the French government's information systems, suggest using a model of six layers interoperability *(Direction générale de la modernisation de l'État (DGME), 2009)*. Alternatively, the Alliance for Internet Of Things Innovation (AIOTI) proposes an interoperability framework of eight layers *(Van der Veer and Wiles, 2008)*. In this sense, this thesis proposes a stack of interoperability based on the principles of agriculture projects. This research proposes the stack of interoperability of five layers: **technical interoperability**, **interoperability of protocol**, **interoperability of data format**, **interoperability of data**

[1] http://www.interoperability-definition.info/en/

**semantic**, and **interoperability of organizations**. Figure 14 illustrates the stack of interoperability and the relations of its layers. The meaning of each layer is as follows.

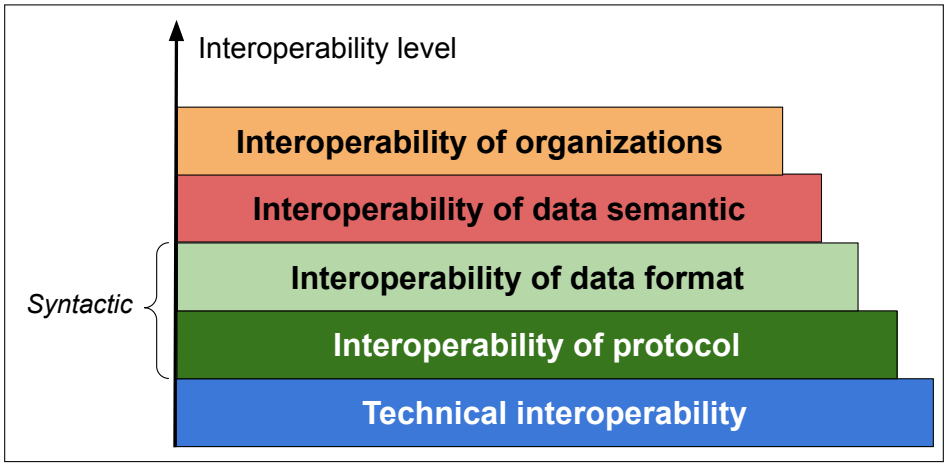



**Figure 14**: Stack of interoperability for CASs

- **Technical interoperability**: Two actors can exchange and process signals. At this level, one actor plays the role of a sender, and the other plays the role of a receiver, or both can be receiver-sender at the same time. The exchange between actors creates physical media, which can be electrical, optical, or wave. The exchange between actors is a physical signal. By using the analog-to-digital converter (ADC) and digital-to-analog converter (DAC), these physical signals can transform to the sequence of bit 0 and 1 that the computer can understand, and inversely computer transforms bits into a physical signal to transfer them over physical media. Electrical signals are transferred through copper cables, for example, cat3, cat5, and coaxial. Optical signals are transferred through optical cables. Wave signals are transferred through wireless or telecommunication technologies such as wifi, Bluetooth, 4G, and 5G.
- **Interoperability of protocol**: Two actors can exchange and process data. Data are encapsulated and decapsulated using a well-known networking structure called the networking protocol. In detail, at the side of the sender, the data will be encoded, divide, and attach to another control data before being sent on physical media at the technical layer. This process itself is divided into many functional layers; for example, in TCP/IP network protocols stack, there are four different layers. Inversely, at the side of the receiver, the raw data will be reformulated by the same functional layers.
- **Interoperability of data format**: Two actors can exchange and process information. The exchanged information between two actors can be read by

both the sender and receiver when they share in a structured format that satisfied the three following rules. First, big data is divided into smaller data pieces; for example, a sentence is divided into words. Second, pieces of data are classified by types; for example, words are categorized into nouns and verbs. Third, some markup words called tag supports to recognize and make pieces of data in order. Example of this mechanism is the format XML, HTML, JSON, and RDF.

- **Interoperability of data semantic**: Two actors share the same data model, or their data models have some similar parts. As a consequence, the two actors can not only read the exchanged information but also has knowledge about the meaning of the information and use it correctly. Some well-known techniques used in the interoperability of data semantic are ontology, semantic annotation, and linked data.
- **Interoperability of organizations**: Actors in this layer are organizations in an agriculture project. Two organizations have a protocol to collaborate for the success of the project.

This research considers ontology as the tool to model data of CASs for two reasons. First, ontology is well-known for data semantic enhancement since it provides not only vocabulary and relations between elements of the vocabulary but also the meta-data to describe such elements. Second, the vocabularies of ontology are sharable and able to use by everyone. Sharing data is critical in IoT, then the data schema of a system should be understood by another system. In other words, ontology is a candidate to archive the interoperability of data semantic.

As the definition of *(Studer et al., 1998)*, ontology is an abstract model that model some phenomenon in the world. An ontology must be machine-readable, be accepted, and shared between a group of people or systems. Moreover, an ontology *"may takes a variety of forms"* according to their objective *(Uschold and Jasper, 1999)*. There are some different methods to classify ontologies, for example, based on language expressivity, based on scope or domain *(Roussey et al., 2011b)*. This research considers the classification that distinguishes between data ontology and logical ontology *(Roussey et al., 2011a)* since it clarifies the contributions of ontology in the context life cycle CAS. A data ontology provides a vocabulary to model data recorded in a system. In detail, the vocabulary contains classes, properties, and individuals. More than that, a logical ontology provides rules to support the reasoning over the data. A logical ontology is built based on a relevant data ontology. In the context life cycle of CAS, the data ontology contributes to the modeling phase, and the logical ontology contributes to the analysis phase.

## 2.2 Comparison of SOSA/SSN and SAREF

Ontology communities encourage to reuse the existed ontologies and vocabularies in forming a new one. In this sense, this research also plans to reuses vocabularies from other well-known ontologies and standards; however, it is essential to select only one of them as the core ontology. There are two steps to determine it:

1. Determine a list of requirements of CASs.
2. Select the ontology that best covers the list of requirements.

The core ontology for the CASs should be an ontology for embedded devices and their measurements. There exist several ontologies addressing this topic *(Bendadouche et al., 2012; Gyrard et al., 2016)*. However, this work only has two final candidates: SOSA/SSN and SAREF, according to two criteria: (1) well-known and being used in several projects, and (2) developed and maintained for a long time by a creditable organization.

The ontology SSN*(Compton et al., 2012a)* was proposed by the World Wide Web Consortium (W3C) and has been broadly adopted worldwide. Addressing the omissions of the original version of the ontology SSN, the joint W3C and Open Geospatial Consortium (OGC) Spatial Data on the Web Working Group has developed a new version of SSN[2] including a module called SOSA[3] (Sensor, Observation, Sampler, and Actuator). This new version, called SOSA/SSN, extends the SSO Pattern (Stimulus Sensor Observation Pattern), implemented in the previous version, by including classes and properties for actuators and sampling. The three major components of SOSA are `sensors` and `observations`, `samplings` and `samples`, and `actuators` and `actuations`.

SAREF[4] is a reference ontology for smart appliances that focuses on the smart homes, and provides a significant contribution to enable semantic interoperability in the IoT being adopted by European Telecommunication Standardization Institute (ETSI) as a Technical Specification *(ETSI, 2015)*. This ontology provides a core model for IoT that could be extended and adapted in order to cover specific domains. SAREF focuses on the representation of `appliances and devices`, together with their `functions, commands, services, states,` and `profiles`. In the latest version, this ontology considers the representation of `measurements` from sensors.

This section compares SOSA/SSN and SAREF to select one as the core ontology. The selection is based on the two steps to determine the core ontology. Section 2.2.1

[2] http://www.w3.org/ns/ssn

[3] http://www.w3.org/ns/sosa

[4] http://www.w3id.org/saref

describes the first step, that is, to determine the list of requirements. Section 2.2.2 presents the second step, that is, select the ontology that best covers the requirements.

### 2.2.1 Requirements Extraction

Requirements to be modeled of a system should be extracted from sources of information related to this system. There are three sources of information as follows.

- Experiences of the computer scientists of the project.
- Experiment dataset of an irrigation CAS maintained by TSCF.
- Books and documents about the standards and concepts of CASs.

After analyzing the above sources of information, the result is a list of requirements. Each requirement is coded as R with a number and the title of the requirement. R means `requirement`, and a number corresponds to the order of a requirement in the list. The list of requirements is as follows.

**R1. Device:** Computing devices and other tools used in a CAS. The CAS irrigation system includes four following types: (1) sensor devices: weather station, tensiometer, and pluviometer; (2) actuator devices: watering sprinkler; (3) gateways; (4) server: a local server based on Linux that supports part of the DSS. This requirement includes four sub-requirements:

**R1.1. Sensor device:** Generic sensor devices.

**R1.2. Actuator device:** Generic actuator devices.

**R1.3. System componency:** Componency relation between devices.

**R1.4. Domain specific device:** Agricultural domain oriented devices, for example, a pluviometer.

**R2. Observation:** A measurement, aggregation, or deduction made by computing devices, including the data about the result of the observation, units of measures, and time-related information about the observation. The following units need to be represented in this agriculture use case: millimeter (mm), centibar (cbar), Watermark unit of measure (the Watermark soil moisture measure ranges from 0 to 200), which is transformed to cbar, and day counting (day). Also, Celsius degrees (°C), Decibel-milliwatts (dBm), and Millivolt (mV) are usually needed in a broader scenario of irrigation systems. This requirement includes four sub-requirements:

**R2.1. Aggregation:** Observations of the calculations made by software programs of computing devices.

**R2.2. Deduction:** Deductions made by an inference engine.

**R2.3. Domain specific result:** Concrete results needed for this agriculture use case; for example, the state of crop.

**R2.4. Domain specific unit:** Concrete units needed for this agriculture use case; for example, the Watermark specific units.

**R3. Property:** Specific agricultural oriented properties as well properties also used in other domains. The following list of properties corresponds to this agriculture use case, and it does not intend to be exhaustive in the agricultural domain: soil moisture, water received during watering, temperature of the soil, temperature of the air, ambient humidity, precipitation, and crop growth stage.

**R3.1. Domain specific property:** Concrete properties to be observed or acted upon needed for this agriculture use case; for example, the soil moisture or the water received by the plot during the watering activity.

**R4. Feature of interest:** The entity that has the property being observed. For example, when measuring temperature, one might distinguish whether it is the temperature of a room or a person. This entity is usually referred to as the feature of interest. This requirement could be further specified as follows:

**R4.1. Domain specific feature of interest:** Concrete feature of interest that has the property being observed for this agriculture use case; for example, the soil.

**R5. Action:** The action that an actuator device performs. This requirement could be further specified as follows:

**R5.1. Domain specific action:** The action `watering` in this agriculture use case. It includes the parameters such as the time interval during which the action has to be carried out.

### 2.2.2 Ontology Requirements Coverage

Table 2 presents the relation between the requirements and the ontology elements defined in the ontologies SOSA/SSN and SAREF. In this table, the first column of a row represents the requirement. The second column and third column in the same row represent the coverage by the SOSA/SSN and SAREF ontologies. They can be in the three following situations. First, when the cell is empty, it means that the given ontology has no vocabulary to cover the requirement. Second, when the cell is covered, the elements from the given ontology are included in the cell (note that these elements might be classes or properties). Third, when there is a number enclosed by parenthesis, the document of the given ontology provides side notes that can be useful for the requirement.

**Table 2**: Requirements coverage by SOSA/SSN and SAREF

| Requirement | SOSA/SSN | SAREF |
|---|---|---|
| **R1 Device** | ssn:System | saref:Device |
| R1.1 Sensor device | sosa:Sensor | saref:Sensor |
| R1.2 Actuator device | sosa:Actuator | saref:Actuator |
| R1.3 System componency | ssn:hasSubSystem | saref:consistsOf |
| R1.4 Domain specific device | | |
| **R2 Observation** | sosa:Observation (1) | saref:Measurement saref:UnitOfMeasure (2) |
| R2.1 Aggregation | (3) | |
| R2.2 Deduction | | |
| R2.3 Domain specific result | | |
| R2.4 Domain specific unit | | |
| **R3 Property** | ssn:Property | saref:Property |
| R3.1 Domain specific property | | |
| **R4 Feature of interest** | sosa:FeatureOfInterest | |
| R4.1 Domain specific feature of interest | | |
| **R5 Action** | sosa:Procedure | saref:Function saref:Command |
| R5.1 Domain specific action | | |

As shown in Table 2, regarding R5, SOSA/SSN documentation proposes to link to the Quantities, Units, Dimensions and Data Types Ontologies (QUDT) *(Hodgson et al., 2014)*, the Ontology of Units of Measure (OM) *(Rijgersberg et al., 2013)*, or the RDF extension mechanism for UCUM (Unified Code for Units of Measure) datatype *(Lefrançois and Zimmermann, 2016)* (Note (1)). For the case of SAREF, the OM ontology is used as a suggestion for covering this aspect (Note (2)). The definition of **sosa:Observation** is quite general in the document of SOSA/SSN. It can be used as observations for both measurement and aggregation; however, in this sense, the data model might be unclear (Note (3)).

Before selecting the core ontology between SOSA/SSN and SAREF, it is

necessary to consider two remarks as follows.

1. The two ontologies cannot cover the domain specifics requirements since they define only general concepts. In this sense, it is not a criticism about the lack of coverage of requirements R1.4, R2.4, R3.1 and R5.1 as they refer to domain-specific knowledge.
2. The vocabularies provided by both of the two ontology cannot clarify different computations: measurement, aggregation, and deduction. Moreover, there is no vocabulary indicated to the qualitative data such as a state.

From these remarks, this research selects SOSA/SSN as the core ontology since it better covers the requirements.

## 2.3 CASO: an Ontology Dedicated to Context-Aware Systems

This research proposes a new ontology for CASs called Context-Aware Systems Ontology (CASO). CASO addresses the need for having a vocabulary that covers the basic requirements of CASs in general. These basic requirements are the different types of computations (measurement, aggregation, and deduction) and their data.

Some ontologies and particular ontology elements were identified to be reused in CASO. Note that most of the selected ontologies have been developed and maintained by standardization organizations.

- The W3C & OGC SSN is the core ontology for CASO and IRRIG. It describes sensors, observations, samples and actuators *(Janowicz et al., 2019)*.
- SAREF is an ontology for smart appliances that contributes to semantic interoperability in the IoT domain *(ETSI TS 103 264 - v2.1.1, 2017)*. This ETSI recommendation focuses on the representation of appliances and devices together with their functions, commands, services, states and profiles (It is necessary to mention that CASO and IRRIG reuse some concepts from SAREF and SAREF4AGRI, which are unavailable in the latest published versions of these ontologies, but it is known by the authors that they will be available in the next versions. SAREF4AGRI is an extension of SAREF for agriculture.).
- The W3C PROV is an ontology that represents provenance information *(Lebo et al., 2013)*.
- The W3C SKOS is a vocabulary that describes knowledge organization systems. It will be useful in defining properties and their states *(Bechhofer and Miles, 2009)*.

- The W3C OWL-Time is an ontology of temporal entities, describing the temporal properties of resources.
- Another well-known ontology reused in this conceptualization is the Ontology of Units of Measure and Related Concepts (OM) *(Rijgersberg et al., 2013)*.

Table 3 contains the prefixes and links to access the above ontologies.

**Table 3**: List of prefixes for CASO and IRRIG

| Prefix | Full address |
|---|---|
| caso | https://w3id.org/def/caso# |
| dc | http://purl.org/dc/elements/1.1/ |
| irrig | https://w3id.org/def/irrig# |
| om | http://www.ontology-of-units-of-measure.org/resource/om-2/ |
| owl | http://www.w3.org/2002/07/owl/ |
| prov | http://www.w3.org/ns/prov# |
| saref | https://w3id.org/def/saref# |
| saref4agri | https://w3id.org/def/saref4agri |
| skos | http://www.w3.org/2008/05/skos# |
| sosa | http://www.w3.org/ns/sosa/ |
| ssn | http://www.w3.org/ns/ssn/ |
| time | http://www.w3.org/2006/time# |
| vann | http://purl.org/vocab/vann/ |

In some cases, the reused ontology elements matched the intended use in the developed ontology; therefore, they were adopted without modification. Some examples of this case are the classes **sosa:Platform**, **sosa:FeatureOfInterest**, or **om:Unit**. However, in other cases, some additional properties or constraints had to be added to the reused elements. Then, a new subclass of reused concepts was created to attach the new semantics to the subclasses created. For example, the case of **caso:Observation** is a subclass of **sosa:Observation**.

The last version of CASO (v201912) is illustrated in Figure 15. The entities defined in CASO are preceded by the prefix **caso** (yellow boxes). It contains a total of 27 classes, 40 object properties, and four datatype properties. CASO has three advantages specialized for CAS as follows.

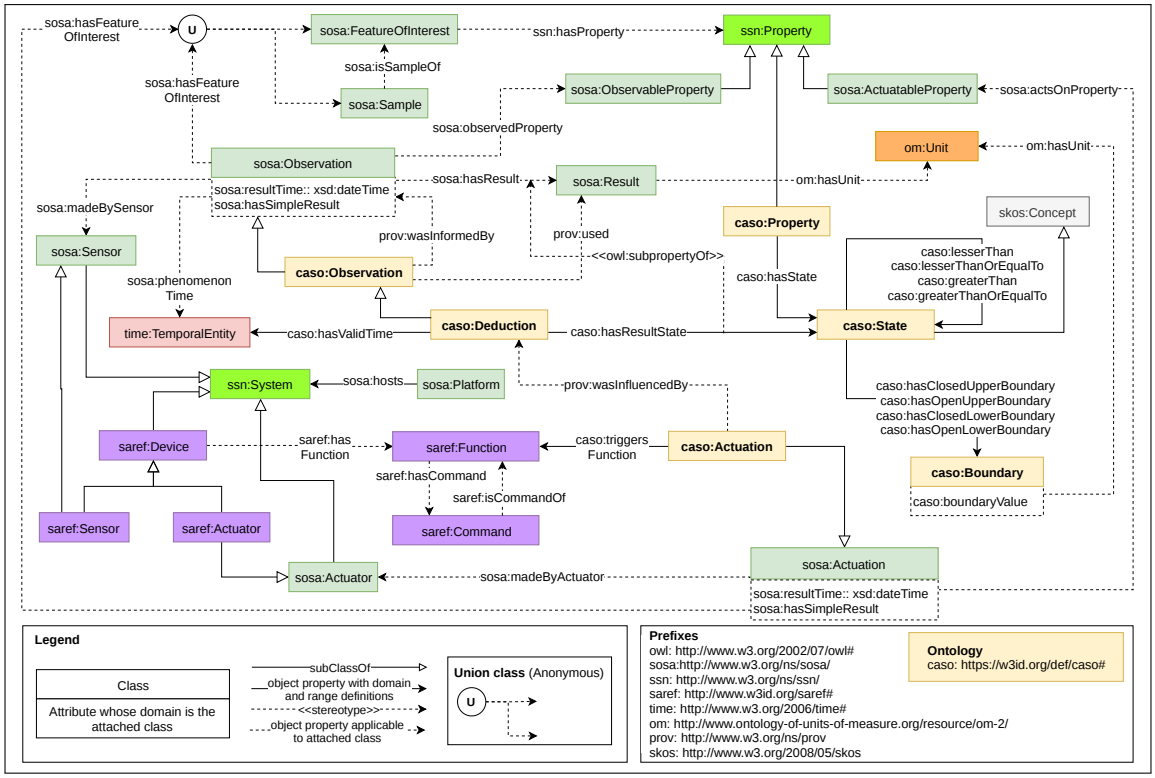



**Figure 15**: Overview of CASO

- The new class **caso:Observation** describes better an aggregation than the class **sosa:Observation**. Indeed, an aggregation, in general, often takes inputs for its calculation. The definition of **sosa:Observation** provides no vocabulary to describe this fact. However, **caso:Observation** uses the `prov:wasInformedBy` object property to identify the inputs of the computation.
- The new class **caso:Deduction** describes better an deduction than **sosa:Observation** and **caso:Observation**. Indeed, the output of a deduction should be data in high-level context which is a state. **caso:Deduction** uses the `caso:hasResultState` object property to present the output of the computation as a state.
- The new class **caso:State** describes states of an entity. The value of a state is defined by using boundaries represented by the new class **caso:Boundary**. The use of **caso:State** and **caso:Boundary** allows understanding the link between the numeric values of an entity and its state. Note that the relations of **caso:Deduction**, **caso:Observation**, **caso:State**, and **caso:Boundary** embody a mechanism to make generic rules for reasoning. In detail, a state can be defined with a range of values. When the observed data is in the range, the deduction can deduct the corresponding state.

CASO is encoded in OWL *(W3C OWL Working Group, 2012)* using Protégé *(Musen, 2015)*. The implementation of the ontologies included the declaration of metadata, such as authors, dates, and licenses, according to Garijo and Poveda Villalon (2017).

## 2.4 Evaluation of CASO

To check bad practices and common errors, the online pitfall scanner OOPS! *(Poveda-Villalón et al., 2014)* has been applied to CASO. This tool is available on the site http://oops.linkeddata.es/.

Figure 16 shows the report after evaluating CASO using OOPS!. In this report, CASO has the pitfalls P10, P11, and P13 since it imports SOSA/SSN. In detail, SOSA/SSN is a generic ontology, so that the restrictions of its classes and properties are loosening. For example, instead of defining the domain and range of properties, it uses `domain include` and `range include`. The reason for the pitfall P04 is that CASO uses different vocabularies from several sources, then some of them are independent of the others. The pitfall P32 is the case of deliberately using the same label, for example, the classes **caso:Observation** and **sosa:Observation**.

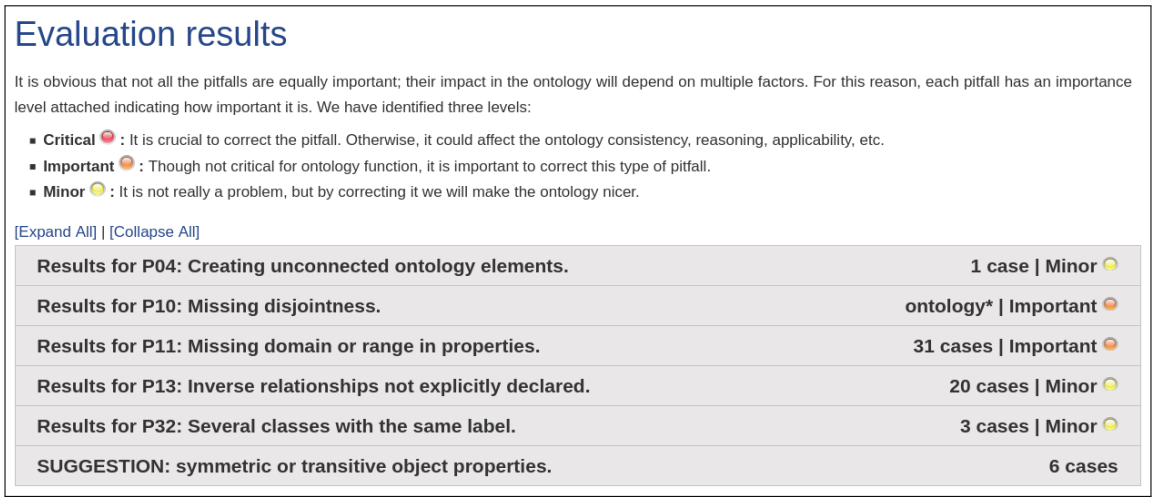



**Figure 16**: Evaluation report of CASO using OPPS!

Publishing a ontology and receiving feedback from users is a solution for the development team to detect and correct errors in a passive way. The publication of the ontology CASO relies on the https://w3id.org/ permanent identifiers initiative. The ontology publication follows the content negotiation mechanism. CASO is identified with the URI https://w3id.org/def/caso#. The content negotiation mechanism deploys to publish the ontologies are transparent to ontology developers as it is

managed by OnToology *(Alobaid et al., 2019)*. OnToology[5] is a web-based system that builds on top of Git-based environments and integrates a set of existing tools for documentation, evaluation and publication activities. The OnToology's integrated version of Widoco*(Garijo, 2017)* generates the HTML published and allows users to update the ontology with new information and diagrams.

The issue tracker provided by the GitHub is the tool to receive feedback and control issue lists. The issue tracker of CASO is available at https://github.com/Irstea/caso/issues. Also, ontology developers provide maintainer email addresses that enable another method for the community to contact. Figure 17 shows the screenshot of the issue tracker for CASO in GitHub.

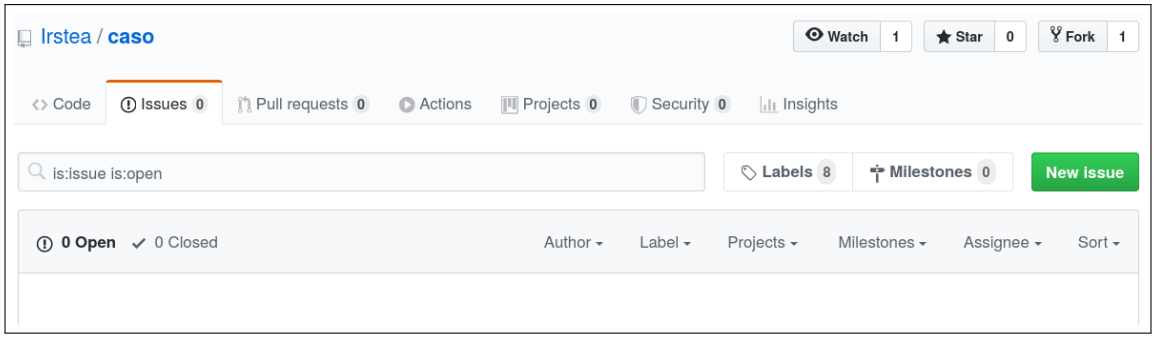



**Figure 17**: Issue tracker interface of the CASO GitHub repository

By publishing an ontology online, it is easy to receive feedback from users. The users can contribute to the development of CASO by giving their idea or by announcing the errors that they detected during usage. The developer team records and verify the feedback before updating them in CASO.

## 2.5 Summary and Discussion

To sum up, this chapter presents CASO, a new ontology dedicated to CASs. This ontology provides vocabularies to model the three types of computation (measurement, aggregation, and deduction) and three types of data (measured data, aggregate data, and deducted data). Also, il includes other concepts related to computations, such as properties and features of interest. Moreover, one remarkable point of the ontology is that it provides a vocabulary to making generic rules for reasoning. CASO reuses several existed ontologies and standards, of which SOSA/SSN is its core. After passing the necessary tests, this ontology is available

[5] http://ontoology.linkeddata.es/

online, and the ontology developers maintain and update the ontology using the issue tracker on GitHub.

It is possible to conclude that this chapter success in responding to the challenge of the interoperability of CASs. However, there are still three points to discuss further:

- The current version (v201912) of the ontology CASO has enough vocabulary to describe measurements, aggregations, and deductions, then the relation between measured data, aggregate data, and deducted data are well connected. However, after the link between deductions and actuations, the relation from deducted data to commands of actuators is still open. This remark could be important, especially for the CAS with the complex actuator.
- This ontology cannot represent the other aspects of CASs, such as the networking and the coordinates of a CAS. It could be a remark to improve CASO in the future.
- CASO is an ontology that can resolve the issue of data heterogeneity in CAS. This ontology is generic for every CAS; thus, its vocabulary is not specialized enough to describe data in specific use cases. For example, a smart home system needs the vocabulary to specify humidity in a room, while a smart irrigation system needs the vocabulary to specify soil moisture.

# Chapter 3

# Development of an Irrigation Decision Support System

*"Start small, think big. Don't worry about too many things at once. Take a handful of simple things to begin with, and then progress to more complex ones. Think about not just tomorrow, but the future. Put a ding in the universe."* – **Steve Jobs**

---

The stack of services for CASs and the ontology CASO are two new contributions to improve the quality of CASs, respectively, in upgradability and interoperability. By using the stack of services for CASs, system developers are free in choosing computing devices and software components as long as the goals of services are guaranteed. Consequently, they can upgrade the system without modifying its functionality. By using CASO, different CASs have a common vocabulary in their data model; thus, it eases data and information sharing. These two new contributions are generic for every CAS. In practice, each specific case study needs to consider its circumstance to use them properly.

This chapter presents a case study of using the stack of services for CASs and the ontology CASO. This case study is to develop an irrigation DSS that automize the irrigation method IRRINOV® to suggest farmers watering their field daily. Technically speaking, it is an expert system composed of three main elements: a knowledge base, a rules base, and an inference engine. The DSS is a part of the irrigation CAS in TSCF. Following the stack of services for CASs, this irrigation CAS is composed of several services. Consequently, The DSS runs some services of this stack of services. In data modeling, this system develops a new ontology called IRRIG. It is a specialization of CASO for this case study.

This chapter is organized as follows. Section 3.1 describes the irrigation method IRRINOV®. Next, Section 3.2 introduces the methodology to develop the irrigation DSS. Section 3.3, Section 3.4, Section 3.5, and Section 3.6 presents respectively the four steps to develop the DSS. Finally, Section 3.7 sums up the chapter and open a discussion.

## 3.1 Method IRRINOV®

The method IRRINOV®[1] is developed by Arvalis and its partners. It proposes a guide for farmers to make irrigation decisions based on measurements of tensiometers and pluviometers. It is designed to answer the following questions.

- When should irrigation be started, or when should watering devices be installed on the plot?
- When should we start each watering cycle?
- When should irrigation be stopped, or when should farmers withdraw the watering devices?

Note that watering cycle means an irrigation system performs a watering activity on all the plots engaged in the system *(Nguyen et al., 2020b)*. Watering cycle duration is the number of days between two consecutive waterings of the same plot.

The method IRRINOV® provides a set of decision tables and recommendations that allow farmers to manage their irrigation system on a single plot. This method proposes numerous variants depending on the soil, plot and crop types. This work uses the method IRRINOV® for the region Midi-Pyrénée, which is dedicated to maize crop plants on clay-limestone soil *(Arvalis et al., 2007)*. Following the IRRINOV® guidelines, the measuring equipment includes the following:

- An IRRINOV® measuring station composed of six Watermark[2] probes to measure the soil water tension. Three Watermark probes should be placed at 30 cm depth in the soil, and the other three should be placed at 60 cm depth in the soil. Figure 18 illustrates the prototype of the IRRINOV® station.
- A mobile pluviometer to measure the amount of water received by the crop during watering.
- A weather station with a pluviometer to measure the quantity of water received by the crop during rainfall.

[1] http://www.irrinov.arvalisinstitutduvegetal.fr/irrinov.asp

[2] https://www.irrometer.com/sensors.html#wm

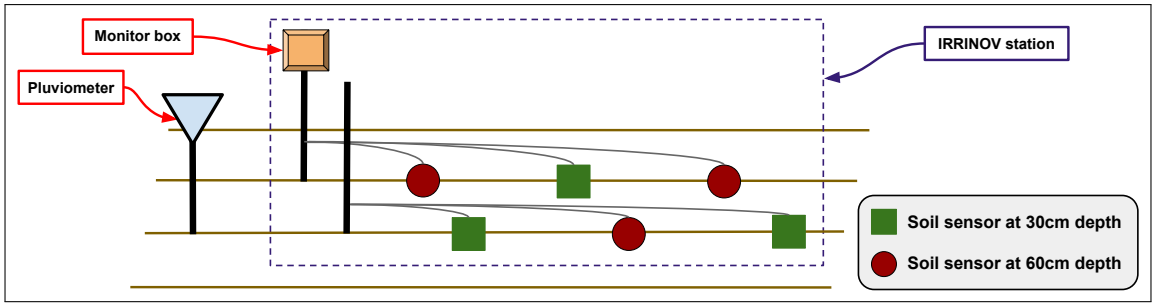



**Figure 18**: Prototype of the IRRINOV® station

The method IRRINOV® specifies the time needed to install the devices on the plot. The IRRINOV® station and the mobile pluviometer should be placed in the plot when the crop reaches growth stage V2[3]. The measurements can start two or three days after installation. The Watermark probes are typically read once a week, but during dry weather, farmers can check the probes more frequently, for example, one observation per day. In addition, Watermark probe measurements should also be carried out as follows:

- Before each planned watering cycle, to confirm or cancel the beginning of a new watering cycle.
- Approximately 24 hours to 36 hours after each watering, to evaluate the effectiveness of the watering. Note that it is necessary to avoid measurements less than 24 hours after watering because they are unstable.
- After significant rains, to evaluate the effects of the rains. For example, if the rainfall amount is under 10 millimeters (mm), then the rain has no impact on the watering cycle plan.

Irrigation should stop when the crop reaches the growth stage R5hg45. Note that state R5 is defined in *(Lori J. Abendroth et al., 2011)*; however, the state R5hg45 is defined in this project. This state means that the crops have moisture dents equivalent to 45%.

The method IRRINOV® specifies the constraints for using Watermark probes. To validate the measurements of Watermark probes, the method IRRINOV® establishes that the difference between the values obtained from different probes at the same depth should be at most 30 cbar. If the difference between the probe measurements is above 30 cbar, then one of the probes is malfunctioning, and the farmer should visit the field to correct the probe installation.

[3]V2, V7, R1, and R5 are the code names of maize growth stages defined in *(Lori J. Abendroth et al., 2011)*. They are respectively named "7 leaf", "10 leaf", "silking" and "dent stage with 50% moisture" in the Arvalis growth stage classification.

The value read on a Watermark probe must be multiplied by the correction coefficient, which is specific for each batch of probes. For example, probes from 2013 have a correction coefficient equal to 1.0. In this paper, we call the former value a raw value. The latter value after this process is the value of the measurement. It represents the soil tension and has the cbar unit.

A difference of 10 to 20 cbar in tension between two probes located at the same depth is considered normal. The method IRRINOV® proposes to install three probes per level of depth. The IRRINOV® guidelines suggest that the farmer start the first watering cycle when two probes out of three reach an initial threshold value. The value reached by two probes out of three is the one taken into account by the decision method. Note that when the raw value is 199 cbar, the method IRRINOV® considers that there is a contact problem between the Watermark probe and the soil.

The method IRRINOV® proposes several decision tables to determine when to start a watering cycle depending on the soil moisture and crop growth stage. Table 4 shows an example that determines when to start a watering cycle for clay-limestone soil. This decision table is applied to maize crops when their growth stage is between V2 and V7.

**Table 4**: Decision table of the relation of the watering cycle duration and the soil moisture for maize at growth stage V2

| **Watering cycle duration** | 9 to 10 days | 6 to 8 days | below 5 days |
|---|---|---|---|
| **Median of 30 cm depth probes** | 30 cbar | 50 cbar | 60 cbar |
| **Median of 60 cm depth probes** | 10 cbar | 20 cbar | 20 cbar |
| **Sum of the median of 30 cm depth probes and the median of 60 cm depth probes** | 40 cbar | 70 cbar | 80 cbar |

The second column of the above decision table should be read as *"If the plot has a watering cycle duration fixed between 9 and 10 days"*, and one of the two following conditions is satisfied:

1. *"When two of three probes at 30 cm depth are above the 30 cbar value"* and *"when two of three probes at 60 cm depth are above the 10 cbar value"*.
2. *"When the total (sum of the median of 30 cm depth probes and the median of 60 cm depth probes) is above 40 cbar"*.

Then, a watering cycle should start.

Note that for this particular decision table, the second and third rows are redundant because of the last one: when the median of 30 cm depth probes is above

30 cbar, and the median of 60 cm depth probes is above 10 cbar, then it is certain that the sum of the two values is above 40 cbar.

This project translates all of the decision tables into rules to transform the manual method IRRINOV® into an automatic DSS.

## 3.2 Methodology Combined of LOT and Mini-Waterfall

The development of the ontology IRRIG is a part of the development of the irrigation DSS. Since ontology engineering is a domain distinguished from system engineering, this research proposes a new methodology for both.

Ontology engineering refers to as *"the set of activities that concern the ontology development process, the ontology life cycle, the methods and methodologies for building ontologies, and the tool suites and languages that support them" (Gómez-Pérez et al., 2004)*. Its goal is to form a new ontology following a methodology. There exist several methodologies to develop an ontology such as Cyc method, KACTUS approach, SENSUS-based, METHONTOLOGY, and Linked Open Terms (LOT) methodology. This research adopts LOT methodology[4] as the candidate to build IRRIG due to its success in several creditable projects *(Poveda-Villalón, 2012; García-Castro et al., 2017)*. LOT is flexible and rapid to change since it relies on the Agile practices[5]. This methodology focuses on (1) the reuse of terms (classes, properties and attributes) existing in already published vocabularies or ontologies and (2) the publication of the ontology following the linked data principle. LOT relies on the ontological engineering activities defined in the NeOn methodology when available *(Suárez-Figueroa et al., 2015)*. LOT defines iterations over the following four activities: (1) ontological requirements specification, (2) ontology implementation, (3) ontology publication, and (4) ontology maintenance. Figure 19 illustrates LOT.

System engineering is the art of developing a system. There exist numerous methodologies to develop a system such as feature-driven development (FDD), waterfall, and scrum. System developers of this project choose mini-waterfall as the methodology to develop this irrigation DSS. The mini-waterfall is inspired by the waterfall methodology, which develops a system in five activities: (1) specification, (2) design, (3) implementation, (4) testing, and (5) maintenance *(Muller and Gaertner, 2004)*. The purpose of the mini-waterfall methodology is to repeat the sequence of the

[4] http://lot.linkeddata.es

[5] Agile is a mindset with the principle to use a set of tools such as SCRUM, KANBAN, and FDD to develop software fast and easy to change.

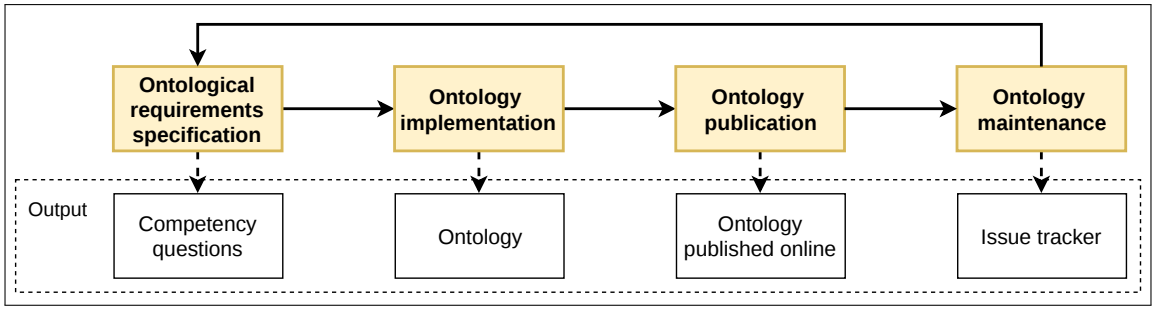



**Figure 19**: Procedure of the LOT methodology

five activities multiple times to develop the system progressively. Figure 20 illustrates the mini-waterfall methodology.

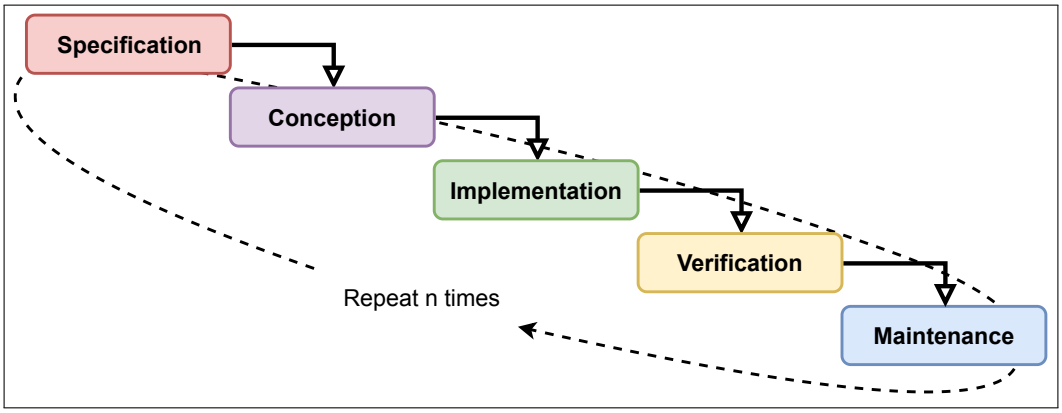



**Figure 20**: Procedure of the mini-waterfall methodology

Figure 21 illustrates the final development methodology for this irrigation CAS. Ontology development has a life cycle called the LOT life cycle. The LOT life cycle is presented by the dashed yellow box. Each activity in this life cycle is represented by a solid yellow box. The solid black arrows between the boxes show the order of the activities; for example, the development team must perform the ontological requirements specification activity before the ontological implementation activity. The CAS development life cycle is called a mini-waterfall life cycle. It is presented by the dashed blue box. Each activity in this life cycle is represented by a solid blue box. The solid yellow arrow between one activity X in the LOT life cycle and one activity Y in a mini-waterfall life cycle means that the result of activity X is necessary to implement activity Y. For example, the design activity of the mini-waterfall cycle requires the result of the ontology conceptualization activity of the LOT life cycle. Otherwise, the dashed blue arrow from one activity Y of mini-waterfall to one activity X of LOT life cycle means that the result of the activity Y can contribute to activity X. It is necessary to note two points in this methodology. First,

the LOT life cycle is triggered by the mini-waterfall life cycle, but it is independent of the mini-waterfall life cycle. As a consequence, the LOT life cycle can be shorter than the mini-waterfall life cycle. Suppose that after each development life cycle, there is a new version. Therefore, there is nothing to prevent the ontology from having three versions at the same time, but the system can only have one version. Second, after the maintenance activities of both the LOT life cycle and CAS life cycle, the development team renews the sequence; that means, the development team starts with the first activity. However, the current activity can be skipped if the last time result of the same activity is sufficient.

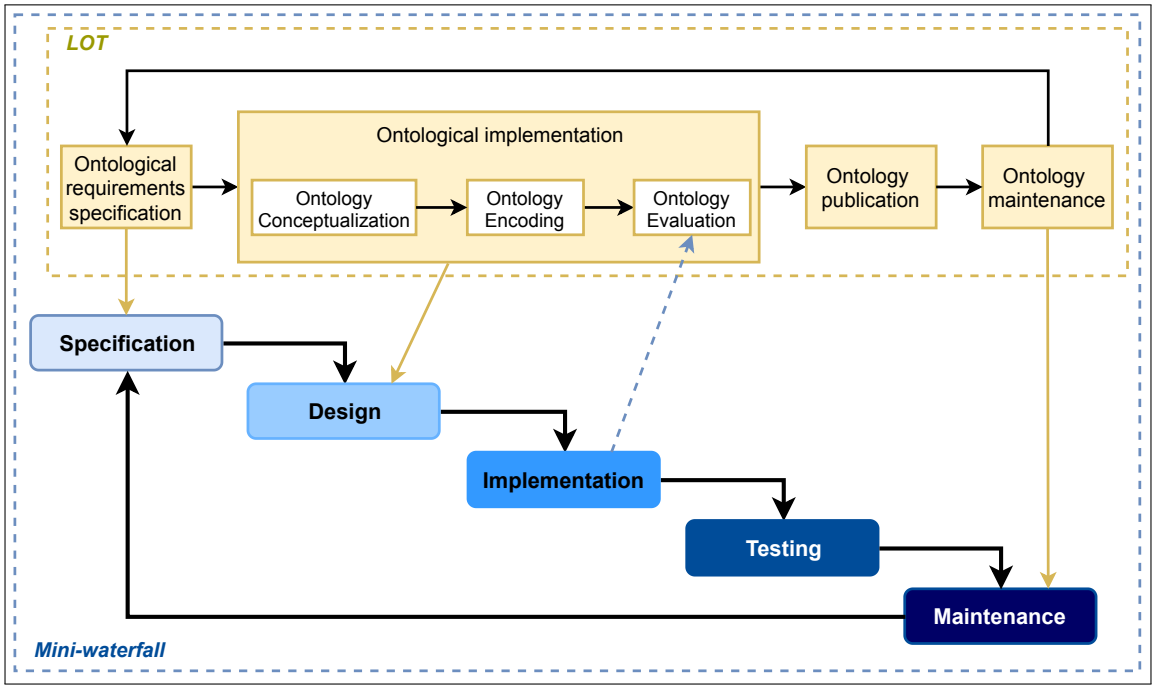



**Figure 21**: Mini-Waterfall combined with LOT methodology to develop a system

## 3.3 System Specification

The system specification step is to analyze the resources required to build the system. In the case study of the irrigation DSS, there are four sources of information as follows.

- The IRRINOV® method version for the region Midi-Pyrénée, France.
- The Arvalis dataset: real experimental data from Arvalis and INRAE. This is an excel file that contains recorded daily observations and activities of farmers in a maize plot T1 B1 - DKC4814 from 14/06/2013 to 06/10/2013 in Gaillac, France. Note that the observations recorded in the Arvalis dataset are the daily

aggregated values of the sensor measurements. In other words, the data in the dataset are already processed from the raw measurements.

- Farming journal ("Carnet de Terrain" in French): a document that contains several tables to record all the precise observations related to the plot. It is a kind of experimentation journal used frequently in research laboratories.
- Experiences and knowledge from specialists in the agriculture domain. The specialists are from Arvalis and INRAE.

The result of the system specification step is a document of competency questions and a system specification document. The competency questions document includes all the questions that the final system should be able to answer. Based on these competency questions, ontology developers can determine the concepts and the relations between the concepts needed for the ontology. Figure 22 presents an excerpt from the current competency questions. A system specification document includes all detail of the system. From such detail, system developers can extract all the functional requirements and non-functional requirements of the system.

**Specification Competency Questions document**

**Version 0.1**

| Code | Compentency Questions | Answer/Example | |
|---|---|---|---|
| CQ1 | Description of measurement equiments | | |
| CQ1.1 | What is the identifier of the probe/sensor? | 2013F13 03 | |
| CQ1.2 | Which type of measurement provides the probe/sensor? Which phenomenon and which associated property is measured by the probe/sensor? | Example: soil moisture at 30 cm-depth | |
| CQ1.3 | What is the correction value of the probe/sensor? Does the raw measurement produced by a probe/sensor need a transformation before being considered as a valid measurement? | 1 | Th va re |
| CQ1.4 | What is the range of value provided by the probe/sensor? Is the measurement of the probe/sensor valid? | 0 to 200 | |
| CQ1.5 | What is the unit of the probe/sensor? | char, mm, degree Celsius | |

**Figure 22**: Screenshot of a part of the competency questions document containing the questions and answers from CQ1.1 to CQ1.4

## 3.4 System Design

The system design step is to transform all the information in the specifications into models. This irrigation DSS has one data model and one system model.

First, the data model has a schema relied on the ontology IRRIG. The data to be modeled are from different computations (observation, aggregation, and deduction) relating to different entities. The entities could be grouped by topic into four workflows: `crop growth`, `rain quantity`, `soil moisture`, and `crop water need`.

Figure 23 shows the four workflows, of which each workflow is a collection of the observations, aggregations, and deductions of entities related to the topic of the workflow. For example, the `crop growth` workflow contains **CropGrowthObservation** and **CropGrowthDeduction**. Observation, aggregation, and deduction produce three types of data: measured data, aggregate data, and deducted data. Measured data are data generated directly from devices. Aggregate data are data calculated through an aggregation process. Deducted data are data inferred after a deduction process. In the figure, a green box represents a measured data, a yellow box represents an aggregate data, and a red box represents a deducted data.

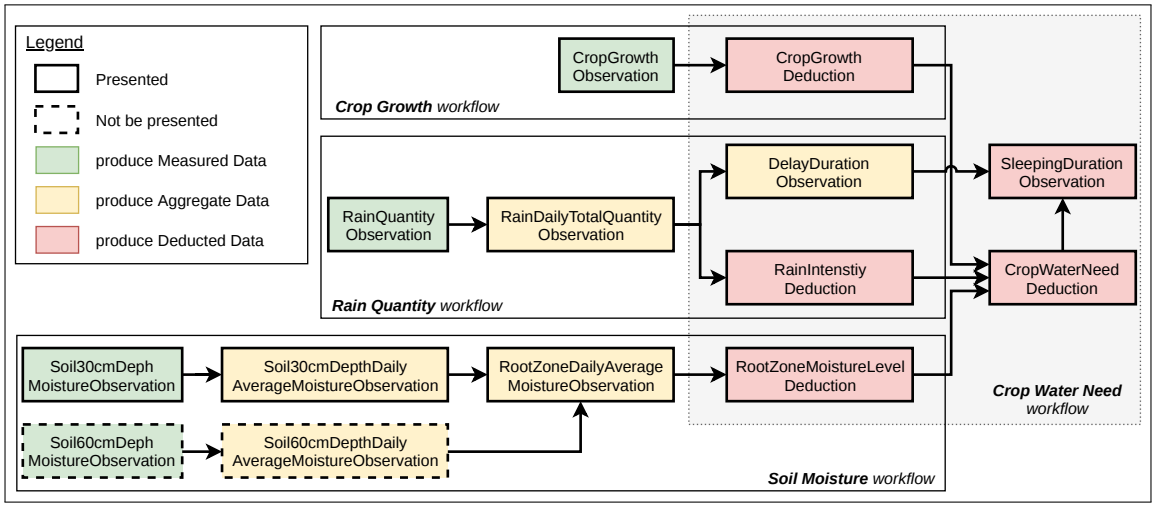



**Figure 23**: Four workflows of the CAS smart irrigation

The system model is a set of services. To recall, the irrigation CAS applies the stack of services for CASs in designing. Since the DSS is only a part of the CAS, its model corresponds to some services in the stack.

The rest of this section is organized as follows. Subsection 3.4.1 describes IRRIG. Subsection 3.4.2 , Subsection 3.4.3, Subsection 3.4.4, and Subsection 3.4.5 presents the data modeling and reasoning of `crop growth workflow`, `rain quantity workflow`, `soil moisture workflow`, and `crop water need workflow` respectively. Finally, Subsection 3.4.6 focuses on the services provided by the DSS.

### 3.4.1 Irrigation Ontology for Experimentation in TSCF

IRRIG is the ontology for the irrigation DSS in TSCF. Figure 24 illustrates the latest version of IRRIG (v201912). It imports and extends CASO with 31 classes, four properties, 71 individuals, and 24 rules. The entities defined in IRRIG are preceded by the prefix **irrig**. IRRIG has three advantages as follows.

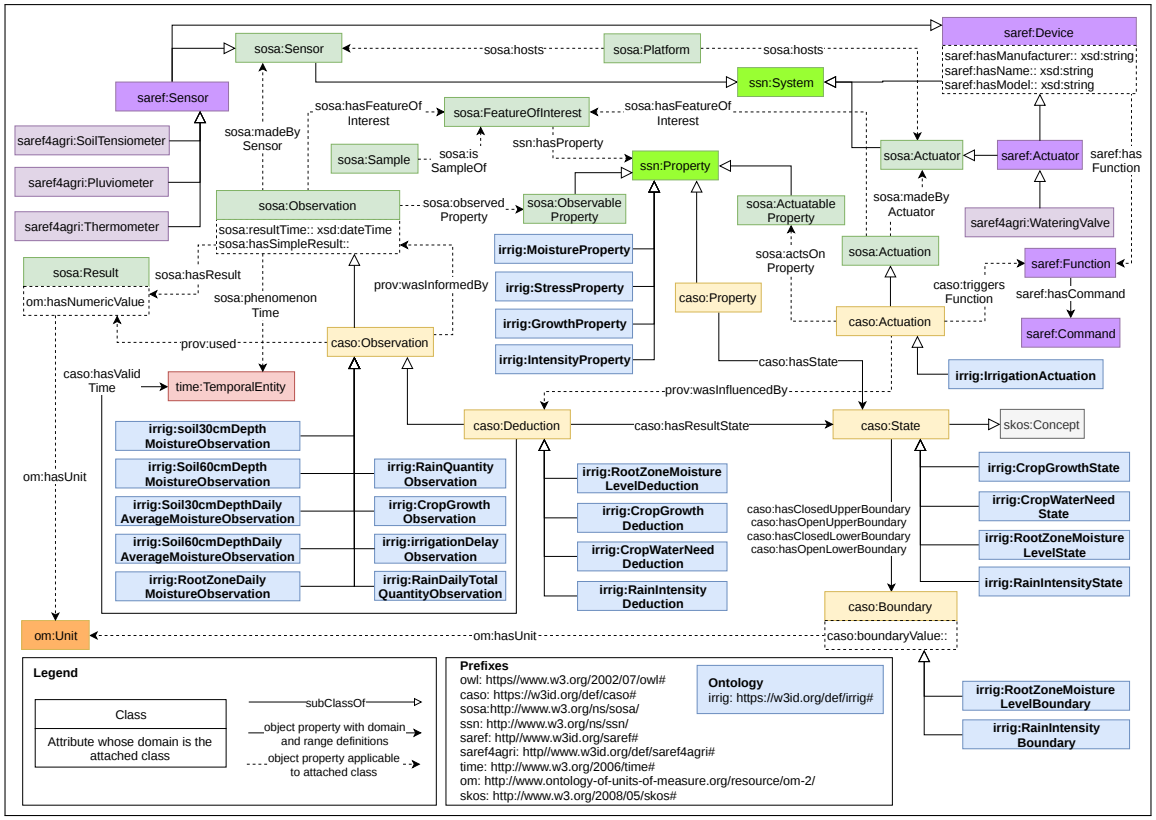



**Figure 24**: Overview of IRRIG

- IRRIG specializes the class **ssn:Property** in different subclasses related to moisture, stress, and growth. Several subclasses of **caso:Observation** specify different types of observations. Each subclass is dedicated to the observations of one specific instance of **ssn:Property**. All of these specializations allows the system to model heterogeneous and numerous data from CASs.
- IRRIG provides 71 individuals related to several entities in agriculture. Each individual is well-defined and is clarified by its metadata. It allows stakeholders and data users to understand the system better.
- The 24 rules of IRRIG allows reasoning on the data. These rules are encoded in Semantic Web Rule Language (SWRL).

### 3.4.2 Crop Growth Workflow Data Modeling and Reasoning

In the `crop growth` workflow, data related to crop growth is processed through several computations. The input of this workflow is the crop growth stage. The output of this workflow is the state of the property CropGrowth. In detail, the `crop growth` workflow composes of two computations as follows.

- **CropGrowthObservation**: The observation provides the growth state of the feature of interest representing the crop. This state is converted from a growth stage, which is observed by a human.
- **CropGrowthDeduction**: The deduction provides the growth state of the feature of interest crop inferred by an inference engine.

#### 3.4.2.1 CropGrowthObservation Modeling

Farmers observe the crop growth stage. These stages represent the life cycle of maize. In the method IRRINOV®, the following stages are observed by farmers:

- Crop growth stage of "10 leaves".
- Crop growth stage of "silking" (flowering begins when a silk is visible outside the husks).
- Crop growth stage of "dent stage with 50% moisture".
- Crop growth stage of "dent stage with 45% moisture".

Table 5 presents the four crop growth observations stored in the Arvalis dataset. Each observation was stored as two datapoints: (1) crop growth stage, and (2) the date of observation (year, month, day). The two first column presents the crop stages, respectively, in French and English. The last column presents the dates of observation.

**Table 5**: Observations of crop growth stage stored in the Arvalis dataset

| Stage in French | Stage in English | Observation date |
|---|---|---|
| 10 feuilles | 10 leaves | 26/06/2013 |
| Floraison femelle | Silking | 27/07/2013 |
| HG50% | Dent with 50% moisture | 10/09/2013 |
| HG45% | Dent with 45% moisture | 19/09/2013 |

The crop growth stages are coded as states for the property CropGrowth. The coding relies on other well-know resources about maize agriculture *(Lori J. Abendroth et al., 2011)*. Moreover, the method IRRINOV® and the irrigation system requires to add some new states. The list of CropGrowth states is as follows:

- **CropGrowth.Init**: The state represents the crop before reaching the "10 leaves" stage. This state is called `Init` because CropGrowth is always at this state right after the initialization of the system. Note that this state also represents all of the stages of the crop until it reaches the "10 leaves" stage.

- **CropGrowth.V7**: The state represents the crop after reaching the "10 leaves" stage, also named `V7` in *(Lori J. Abendroth et al., 2011)*.
- **CropGrowth.V7d20**: The state represents the crop 20 days after reaching the "10 leaves" stage. Note that IRRINOV® method has many versions corresponding to crop specialty and soil. The number of days for this state is different regard on IRRINOV® method versions. In the case of the Arvalis dataset, it is 20 days.
- **CropGrowth.R1**: The state represents the crop after reaching the "silking" stage. This stage is also named `R1` in *(Lori J. Abendroth et al., 2011)*.
- **CropGrowth.R1d15**: The state represents the crop 15 days after reaching the "silking" stage.
- **CropGrowth.R5**: The state represents the crop after reaching the "dent stage within kernel containing 50% moisture". This stage is also named `R5` in *(Lori J. Abendroth et al., 2011)*.
- **CropGrowth.R5hg45**: The state represents the crop after reaching the "dent stage within kernel containing 45% moisture". When the crop reaches this development stage, the irrigation could stop.

To recall, the CropGrowth states of the system could describe the life cycle of maize. They follow a chronology order: `Init` < `V7` < `V7d20` < `R1` < `R1d15` < `R5` < `R5hg45`. In this order, a state occurs before its right state and after its left one. The change of states takes more than one day. For example, `V7d20` always happens 20 days after `V7`, and at least one day before `R1`.

Figure 25 presents the model of the CropGrowth observation on 27/07/2013 using CASO and IRRIG. Note that the data of the CropGrowth observation on 27/07/2013 is a part of the data sample on 14/08/2013 in the Arvalis dataset.

- An instance of the class **irrig:CropGrowthObservation** reperesents the farmer observation, as shown in Figure 25. By definition, instances of **CropGrowthObservation** are linked to the individual irrig:observedProperty_crop_growth, an instance of the class **irrig:GrowthProperty**, and to the individual irrig:featureOfInterest_crop, an instance of the class **sosa:FeatureOfInterest**.
- The observation individual is linked to the actor that made the observation. In this case, this actor is a farmer. This model uses a sensor to represent the farmer. Since Arvalis provides no information about the name of the farmer, the individual arvalis:people_john_doe, an instance of the class **sosa:Sensor**, is used to represent this actor. The `sosa:madeBySensor` object property links the observation individual to the sensor individual, as shown in Figure 25.

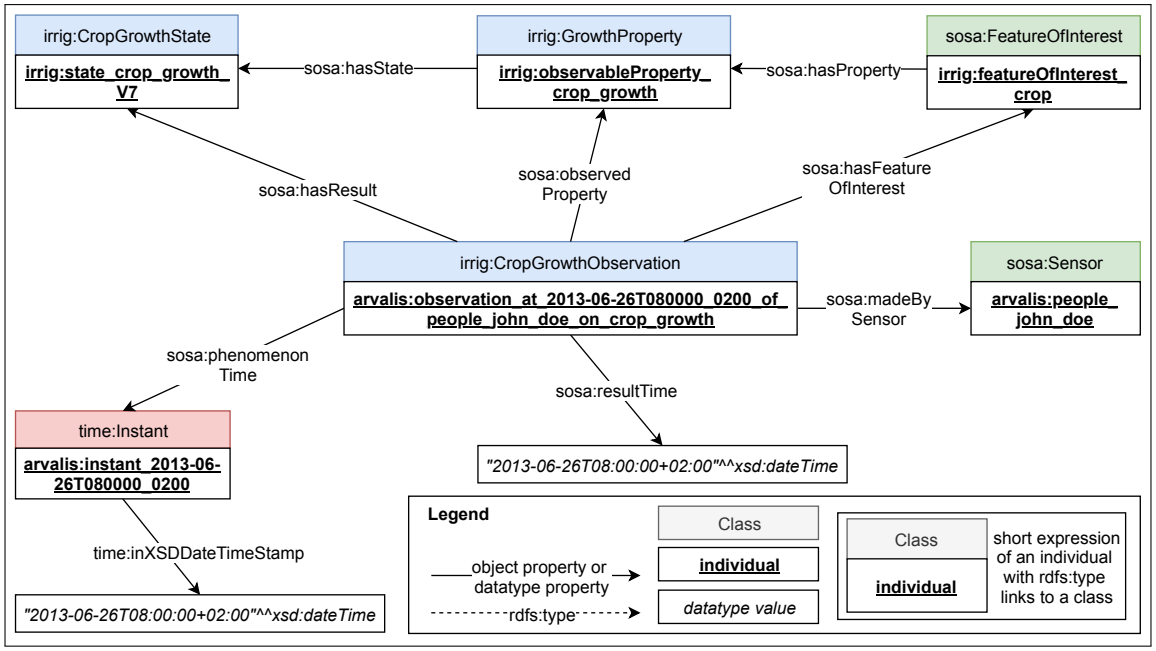



**Figure 25**: CropGrowth observation on 27/07/2013

- Human observation is instantaneous. Farmers made observations at a precise time. Since Arvalis provides no information about the precise hour, minute, and second of observation, it is fixed to 08:00:00 as for the synchronization reason. Figure 25 presents an observation that happens at 08:00:00 on 14/08/2013. To store the temporal information of the measurement, this model uses an instance of the class **time:Instant**. The time:inXSDDateTimeStamp data property stores the associated date-time value. The `sosa:phenomenonTime` object property links the observation individual to the instant individual.
- To present the time when the farmer provides the observation, the sosa:resultTime datatype property is used to link the observation individual to the `xsd:dateTime` value corresponding. In Figure 25, this precise time is at 08:00:00 on 14/08/2013.
- The result of a CropGrowth observation is an instance of the class **irrig:CropGrowthState**. The observation individual is linked to its state result by the `caso:hasResultState` object property, as shown in Figure 25. The possible states are shown in Figure 26.

#### 3.4.2.2 CropGrowthDeduction Modeling

The daily state of the property CropGrowth is deducted based on the last CropGrowth observation. The SWRLAPI Drools Engine runs this deduction process. The last

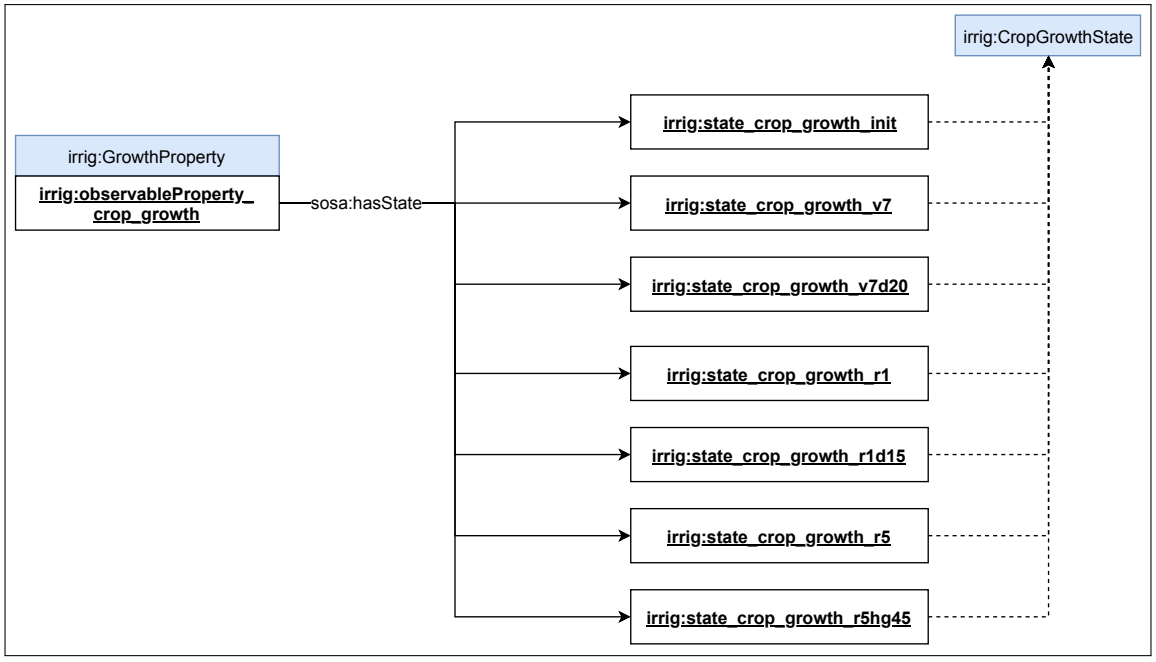



**Figure 26**: States of the property CropGrowth

CropGrowth observation provides two information to deduct the state of CropGrowth of a day: the last state of CropGrowth and the observation day. The second information is to calculate the time interval between the last observation day and the deduction day. By comparing this interval with a temporal threshold, two states can be distinguished. Some temporal thresholds are defined as follows:

- **Th_Time_15D**: The threshold represents a duration of 15 days. It is used to detect the change from the state `R1` to the state `R1d15`.
- **Th_Time_20D**: The threshold represents a duration of 20 days. It is used to detect the change from the state `V7` to the state `V7d20`.
- **Th_Time_1D**: The threshold represents a duration of 1 day. It is used to access the final state.

Figure 27 describes the state diagram that presents the mechanism to deduce the state of the property CropGrowth of a day. In the diagram, there are nine states and eight external transitions. The nine states include two default states of the UML state diagram (initial and final states) and eight states defined for CropGrowth. The variable $t_{crop}$ represents a clock dedicated to CropGrowth that has a time step of one day. The transitions occur when their conditions are satisfied. Each state of CropGrowth in the diagram may contain internal transitions.

- The transition from the initial state towards the state `Init` indicates that as soon as the system starts, the property CropGrowth reaches `Init`.

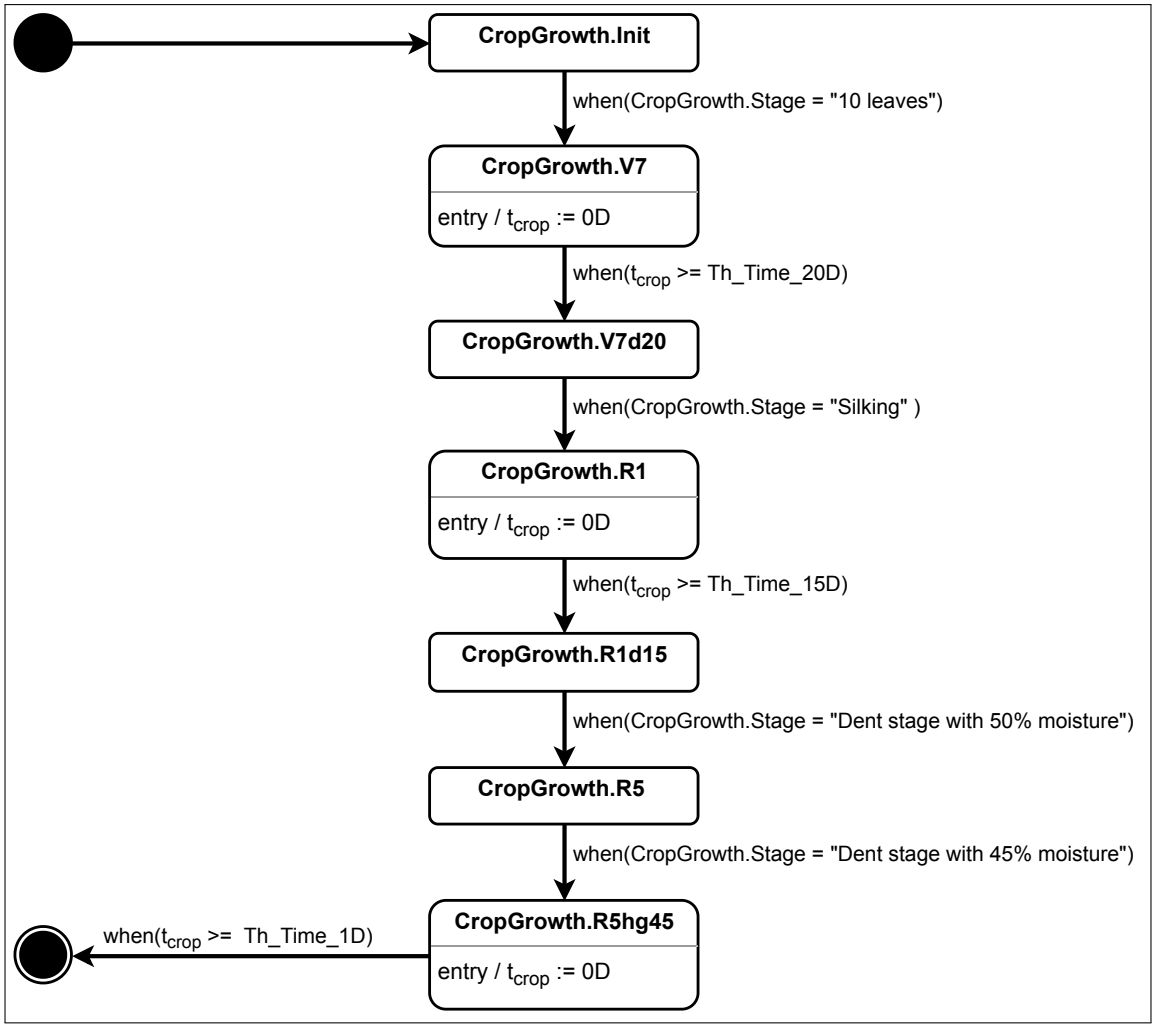



**Figure 27**: Automata of the CropGrowth states and their transitions

- The transition from the state `Init` towards the state `V7` indicates that a farmer observes the maize growth reaching the stage "10 leaves", then the property CropGrowth reaches `V7`. After reaching this state, $t_{crop}$ is reset.
- The transition from the state `V7` towards the state `V7d20` indicates that when $t_{crop}$ is superior or equal to 20 days, the property CropGrowth reaches `V7d20`.
- The transition from the state `V7d20` towards the state `R1` indicates that a farmer observes the maize growth reaching the stage "silking", then the property CropGrowth reaches `R1`. After reaching this state, $t_{crop}$ is reset.
- The transition from the state `R1` towards the state `R1d15` indicates that when $t_{crop}$ is superior or equal to 15 days, the property CropGrowth reaches `R1d15`.
- The transition from the state `R1d15` towards the state `R5` indicates that a

farmer observes the maize growth reaching the stage "dent stage with 50% kernel moisture content", then the CropGrowth property reaches the state `R5`.

- The transition from the state `R5` towards the state `R5hg45` indicates that a farmer observes the maize growth reaching the stage "dent with 45% kernel moisture content", then the property CropGrowth property reaches `R5hg45`. After reaching this state, $t_{crop}$ is reset.
- The transition from the state `R5hg45` towards the final state indicates that when the clock $t_{crop}$ is at least one day, the crop observation campaign is terminated.

Figure 28 and Figure 29 present the model of the CropGrowth observation on 14/08/2013 using CASO and IRRIG.

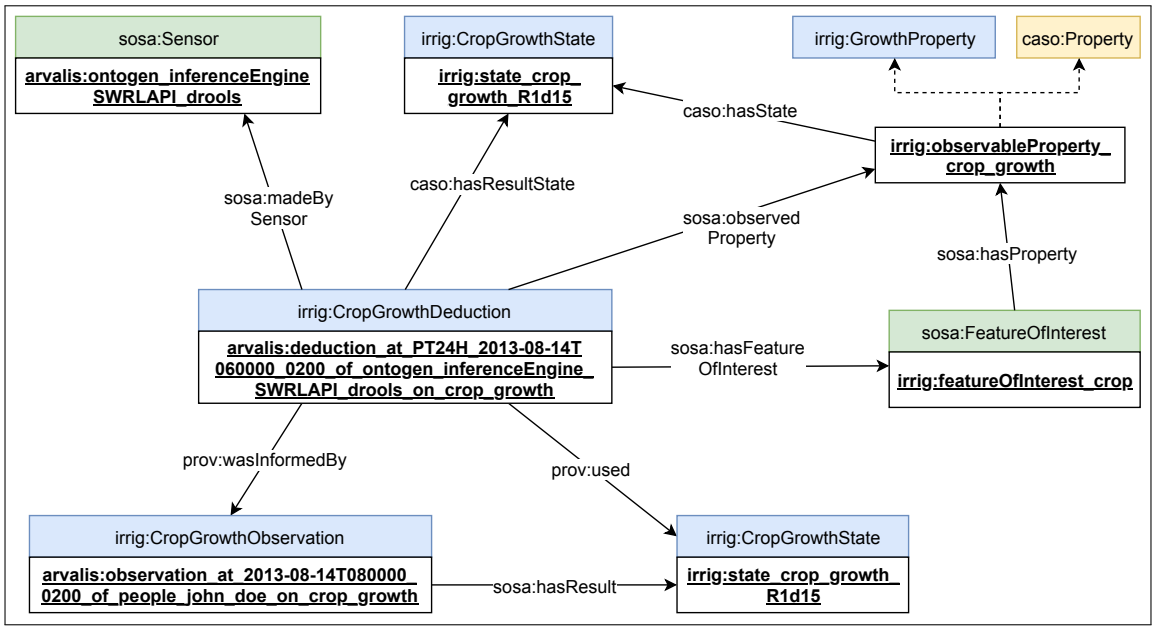



**Figure 28**: CropGrowth deduction on 14/08/2013 (part 1)

- An instance of the class **irrig:CropGrowthDeduction** represents a deduction process for the property CropGrowth property, as shown in Figure 28. By definition, instances of **irrig:CropGrowthDeduction** are linked to the individual irrig:observedProperty_crop_growth and to the individual irrig:featureOfInterest_crop.
- The deduction individual is linked to an actor that produces the result. The actor is the SWRLAPI Drools Engine which is represented by the individual arvalis:ontogen_inferenceEngine_SWRLAPI_drools, an instance of the class **sosa:Sensor**. The deduction individual is linked to the sensor individual via the `sosa:madeBySensor` object property, as shown in Figure 28.

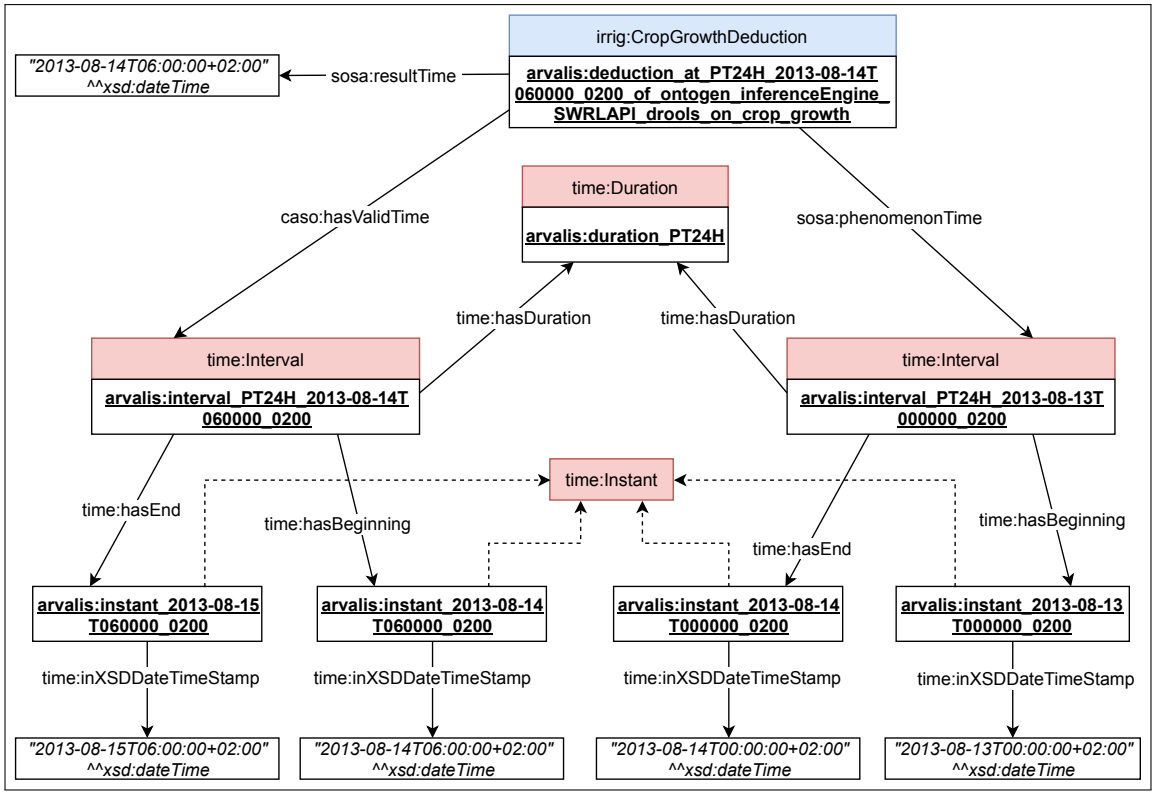



**Figure 29**: CropGrowth deduction on 14/08/2013 (part 2)

- Since a CropGrowth deduction is based on the last CropGrowth observation, the deduction individual in Figure 28 is based on the observation individual presented in Figure 25. The deduction individual is linked to the instance of the class **irrig:CropGrowthObservation** via the `prov:wasInformedBy` object property.
- The result of a CropGrowth deduction is an instance of the class **irrig:CropGrowthState**. The deduction individual is linked to its result by the `caso:hasResultState` object property, as shown in Figure 28. The possible states are shown in Figure 26.
- To store the time of the deduction, the sosa:resultTime datatype property is used to link the deduction individual to the xsd:dateTime value corresponding to the result time at `[d+1 06:00:00]`. In Figure 29, the result time is at 06:00:00 on 15/08/2013.
- To represent the fact that a deduction concerns the data collected on the day d, the object property `sosa:phenomenonTime` is used. In Figure 29, the phenomenon time is represented by an instance of the class **time:Interval** for an interval of `[14/08/2013 00:00:00, 15/08/2013 00:00:00[`, which

is associated with two instances of the class **time:Instant**. The first one represents the beginning of the interval: `[14/08/2013 00:00:00]`. The **time:Interval** instance is linked to the beginning **time:Instant** instance by the `time:hasBeginning` object property. The second one represents the end of the interval: `[15/08/2013 00:00:00]`. The **time:Interval** instance is linked to the ending **time:Instant** instance by the `time:hasEnding` object property. Each **time:Instant** instance uses a time:inXSDDateTimeStamp data property to store the associated date-time value. The `sosa:phenomenonTime` object property links the deduction individual to the interval individual.

- The deduction result for day d is valid for a 24-hour interval: `[d+1 06:00:00, d+2 06:00:00[`. No new deduction will be performed during this valid time. In Figure 29, the valid time is represented by an instance of the class **time:Interval** for an interval of `[15/08/2013 06:00:00, 16/08/2013 06:00:00[`, which is associated with two instances of the class **time:Instant**. The `caso:hasValidTime` object property links the deduction individual to the valid time.

Figure 30 supports to clarify the synchronization of all computations in the `crop growth` workflow.

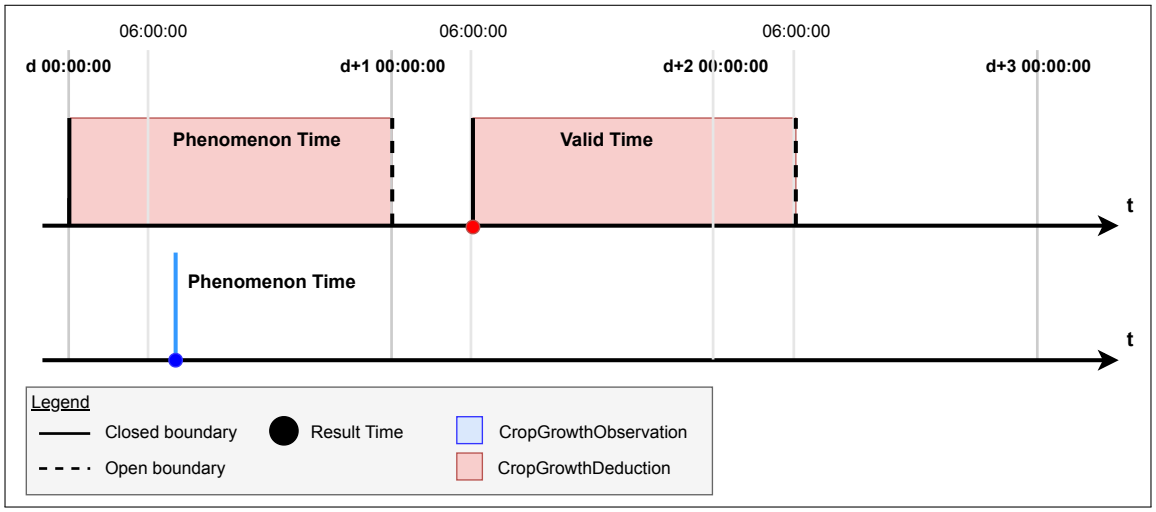



**Figure 30**: Time scales of the crop growth workflow

#### 3.4.2.3 CropGrowthDeduction Reasoning

Figure 27 gives a global view of the mechanism to deduct a CropGrowth state. In reasoning, this mechanism is in the form of one or several rules. There are a total of six rules corresponding to the six transitions towards the states `V7`, `V7d20`, `R1`, `R1d15`,

R5, and R5hg45. All of them are coded in SWRL. While the rules for the states R5 and R5hg45 only need the last CropGrowth observation as input, the other four rules require furthermore to compare the number of day at a state with a threshold. This subsection presents two rules. The first one is the rule for the state R5 representing the former case: it only considers the last CropGrowth observation as input. The second one is the rule for the state V7 representing the latter case.

Table 6 contains necessary information of the rule for the state R5. The short name of the rule is in the box **Code**, and its full name is in the box **Full name**. The description of the rule is in the box **Description**. The code in the box **Rule in SWRL** presents the logic of the reasoning process. Detail of the code is as follows.

**Table 6**: Rule for the property CropGrowth to reach the R5 state

| **Code:**<br>ADv122019-CG-R5 | **Full name:**<br>ArvalisData-IrrigVersion122019-CropGrowth-R5 |
|---|---|
| **Description:**<br>The goal of this rule is to determine the state of CropGrowth. The rule implements the transition from the R1d15 or R5 state to the R5 state. The input of this rule is the farmer observation at the R5 state (?observation_crop_growth). The output of this rule is the Crop Growth deduction at the R5 state (?deduction_crop_growth). This rule's mechanism is to check if the last farmer observation is at R5 state, then the deduction is at the R5 state. | |
| **Rule in SWRL:**<br>irrig:CropGrowthObservation(?observation_crop_growth) ^<br>sosa:hasResult(?observation_crop_growth, ?irrig:state_crop_growth_r5) ^<br><br>irrig:CropGrowthDeduction(?deduction_crop_growth) ^<br>prov:wasInformedBy(?deduction_crop_growth, ?observation_crop_growth) ^<br><br>-> caso:hasResultState(?deduction_crop_growth, irrig:state_crop_growth_r5) | |

- The first paragraph is one premise. It means to find a crop growth observation that has the state R5.
- The second paragraph is one premise. It means to find a crop growth deduction that was informed by a crop growth observation.
- The final paragraph is one deduction. It means a crop growth deduction has a result at the state R5.

Table 7 contains necessary information of the rule for the state V7. The code in the box **Rule in SWRL** presents the logic of the reasoning process. Detail of the code is as follows.

- The first paragraph is one premise. It means to find a crop growth observation

**Table 7**: Rule for the property CropGrowth to reach the V7 state

| **Code:** | **Full name:** |
|---|---|
| ADv122019-CG-V7 | ArvalisData-IrrigVersion122019-CropGrowth-V7 |

**Description:**
The goal of this rule is to determine the state of CropGrowth. The rule implements the transition from the Init or V7 state to one of the V7 state. The input of this rule is the farmer observation at the V7 state (?observation_crop_growth). The output of this rule is the Crop Growth deduction at the V7 state (?deduction_crop_growth). The mechanism of this rule is to check if the duration (?duration_observation_deduction) between the farmer observation and the deduction is inferior to a time threshold (?value_Th_Time_20D) then the result of the deduction is the V7 state.

**Rule in SWRL:**

```
irrig:CropGrowthObservation(?observation_crop_growth) ^
sosa:hasResult(?observation_crop_growth, ?irrig:state_crop_growth_v7) ^

irrig:CropGrowthDeduction(?deduction_crop_growth) ^
prov:wasInformedBy(?deduction_crop_growth, ?observation_crop_growth) ^

time:Instant(?instant_observation) ^
sosa:phenomenonTime(?observation_crop_growth, ?instant_observation) ^
time:inXSDDateTimeStamp(?instant_observation, ?timestamp_observation) ^

time:Interval(?interval_deduction) ^
sosa:phenomenonTime(?deduction_crop_growth, ?interval_deduction) ^
time:hasBeginning(?interval_deduction, ?instant_begin_deduction) ^
time:inXSDDateTimeStamp(?instant_begin_deduction, ?timestamp_begin_deduction) ^

temporal:duration(?duration_observation_deduction, ?timestamp_begin_deduction,
                                                   ?timestamp_observation, "Minutes") ^

caso:hasOpenUpperBoundary(irrig:state_crop_growth_v7, ?Th_Time_20D) ^
caso:boundaryValue(?Th_Time_20D, ?value_Th_Time_20D) ^

swrlb:lessThan(?duration_observation_deduction, ?value_Th_Time_20D)

-> caso:hasResultState(?deduction_crop_growth, irrig:state_crop_growth_v7)
```

that has the state `V7`.

- The second paragraph is one premise. It means to find a crop growth deduction that was informed by a crop growth observation.
- The third paragraph is one premise. It means to get the time value of a crop growth observation.
- The fourth paragraph is one premise. It means to get the time value of the

beginning of a crop growth deduction.

- The fifth paragraph is one premise. It means calculating the interval between the time of crop growth observation and the time of the beginning of a crop growth deduction.
- The sixth paragraph is one premise. It means to get the time value of the threshold equaling to 20 days.
- The seventh paragraph is one premise. It means to check if the time value of the interval is inferior to the time value of the threshold.
- The final paragraph is one deduction. It means a crop growth deduction has a result at the state `V7`.

The rules for `V7d20`, `R1`, `R1d15`, and `R5hg45` have the same logic but different parameters with the two above. The tables of such rules are in the Appendix A.

### 3.4.3 Rain Quantity Workflow Data Modeling and Reasoning

In the `rain quantity` workflow, data related to rain quantity is processed through several computations. The input of this workflow is rain quantity measured during a duration. The output of this workflow is the state of the property RainIntensity and the value of the property DelayDuration. In detail, the `rain quantity` workflow composes of four computations:

- **RainQuantityObservation**: The observation provides the quantity measured during a duration of the feature of interest representing the rain.
- **RainDailyTotalQuantityObservation**: The aggregation provides the total daily quantity of the feature of interest rain. It equals the daily total of the property RainQuantity
- **DelayDurationObservation**: The aggregation provides the duration of the feature of interest delay. The value of the property DelayDuration is calculated based on the value of the property RainDailyTotalQuantity.
- **RainIntensityDeduction**: The deduction that provides the state of the property RainIntensity.

#### 3.4.3.1 RainQuantityObservation Modeling

A pluviometer measures the rain quantity falling during a duration. It depends on the measurement frequency of the pluviometer. The unit of rain quantity is millimeter (mm). For example, the pluviometer measures that the rain quantity falling during one hour is 10 mm. This process is represented by a function

called `getRainQuantity(p,t1,t2)`. This function returns the measurement of the pluviometer p during the interval [t1, t2[.

Figure 31 presents the model of the RainQuantity observation in `[07:00:00, 08:00:00[` on 14/08/2013 using CASO and IRRIG.

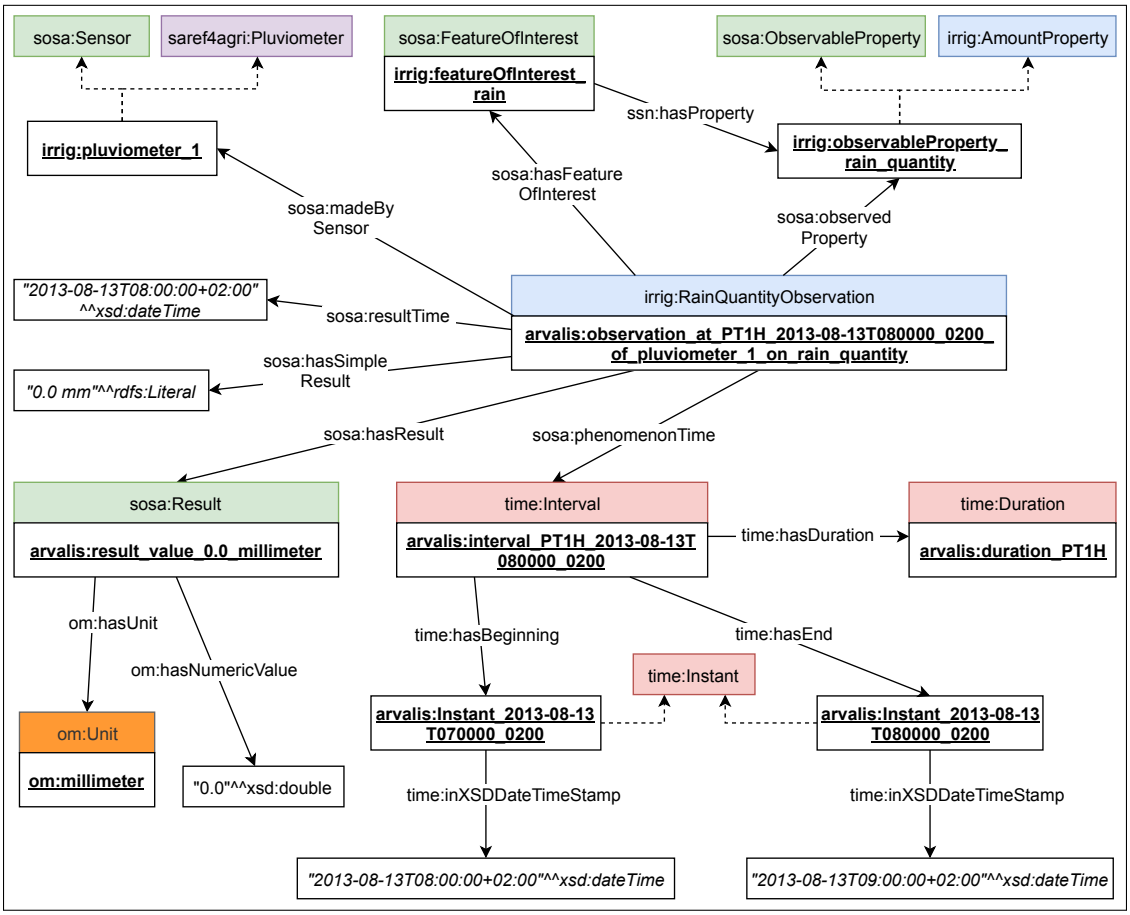



**Figure 31**: RainQuantity observation in `[07:00:00, 08:00:00[` on 14/08/2013

- An instance of the class **irrig:RainQuantityObservation** represents the aggregation performed by running the function `getRainQuantity(p,t1,t2)`. It is an observation of a quantity property related to the rain, as shown in Figure 31. By definition, individuals of **irrig:RainQuantityObservation** are linked to the individual irrig:observedProperty_rain_quantity, an instance of the class **irrig:AmountProperty**, and to the individual irrig:featureOfInterest_rain, an instance of the class **sosa:FeatureOfInterest**.
- The observation individual is linked to the actor that has made the observation, that is, a pluviometer. A pluviometer is represented by an instance of the class **sosa:Sensor**, which is also an instance of the class **saref4agri:Pluviometer**. Note

that in the Arvalis dataset, there is no information about the identification of the pluviometer. Then, the title of the sensor individual is set to arvalis:pluviometer_1, as shown in Figure 31. The `sosa:madeBySensor` object property links the observation individual to the sensor individual.

- A pluviometer measures the RainQuantity property during a time interval. To store the time information of the measurement, this model uses an instance of the class **time:Interval**. In Figure 31, the **time:Interval** instance represents the period of `[07:00:00, 08:00:00[` on 14/08/2013. This instance is described by two instances of the class **time:Instant**. The first one represents the beginning of the interval: `[14/08/2013 07:00:00]`. The object property `time:hasBeginning` links the **time:Interval** instance to the beginning **time:Instant** instance. The second one represents the end of the interval: `[14/08/2013 08:00:00]`. The **time:Interval** instance is linked to the ending **time:Instant** instance by the object property `time:hasEnding`. Each **time:Instant** instance uses a time:inXSDDateTimeStamp data property to store the associated date-time value. The `sosa:phenomenonTime` object property links the observation individual to the interval individual.
- To present the time when the pluviometer generates the measurement, the sosa:resultTime datatype property is used to link the observation individual to the `xsd:dateTime` value corresponding to a precise time. In Figure 31, this precise time is at 08:00:00 on 15/08/2013.
- The result of pluviometer measurement is presented by an instance of the class **sosa:Result**. A value and a unit are required to describe this result. The instance of **sosa:Result** is linked to the data value via the `om:hasNumericValue` datatype property. The instance of the **sosa:Result** is linked to the individual om:millimeter, an instance of **om:Unit**, via the `om:hasUnit` object property, as shown in Figure 31.

#### 3.4.3.2 RainDailyTotalQuantityObservation Modeling

The data recorded in the Arvalis dataset are not the measurements of one pluviometer, but an aggregation of those measurements: the sum of all rain quantity measurements that falls down during a day. The measured data collected from the pluviometer are processed to become the sum of daily total rain quantity. To executing this aggregation process, the function `getRainDailyTotalQuantity(p,d)` is run. Table 8 presents the function `getRainDailyTotalQuantity(p,d)`. This function returns the sum of all pluviometer measurements measured by the pluviometer p during the 24-hour interval. According to Météo-France, the rain quantity of a day is recorded

between two instants [d 06:00:00] and [d+1 06:00:00] [6]. Thus, the 24-hour interval of the day d is represented as `[d 06:00:00, d+1 06:00:00[`.

**Table 8**: Function getRainDailyTotalQuantity(p,d)

$$\boldsymbol{getRainDailyTotalQuantity(p,d) = \sum getRainQuantity(p,t_1,t_2)}$$

on the condition that $[t_1,t_2] \subset$ [d 06:00:00, d+1 06:00:00[

where

- d: a specific day, e.g., 14/08/2013
- p: a pluviometer
- $t_i$: a specific instant of day d, e.g., 08:00:00 on 14/08/2013
- getRainQuantity(p,$t_1$,$t_2$): the rainfall quantity value measured by the pluviometer p during the interval $[t_1,t_2[$.

In the Arvalis dataset, each observation was stored as two data points: (1) the daily total rain quantity in mm, and (2) the date of observation (year, month, day).

Figure 32 presents a sample of the RainDailyTotalQuantity observation on 14/08/2013 to be modeled using CASO and IRRIG.

- The instance of the class **irrig:RainDailyTotalQuantityObservation** represents the computation of the daily total of all quantity measurements produced by one pluviometer, as shown in Figure 32. By definition, instances of **irrig:RainDailyTotalQuantityObservation** are linked to the individual irrig:observedProperty_rain_dailyTotalQuantity, an instance of the class **irrig:AmountProperty**, and to the individual irrig:featureOfInterest_rain.
- The observation individual is linked to the actor that produces the result, that is, an aggregator runs the function `getRainDailyTotalQuantity(p,d)`. Since there is no information about the aggregator that produces this computation in the Arvalis dataset, it is possible to represent it by the individual arvalis:aggregator_getRainDailyTotalQuantity, an instance of the class **sosa:Sensor**. The observation individual is linked to the sensor individual via the `sosa:madeBySensor` object property, as shown in Figure 32.
- The aggregator produces the result once per day at a precise time. To present this time, the sosa:resultTime datatype property is used to link the observation individual to the `xsd:dateTime` value. In the Arvalis dataset, there is no

[6] http://services.meteofrance.com/e-boutique/climatologie/duree-retour-precipitation-journalier-detail.html

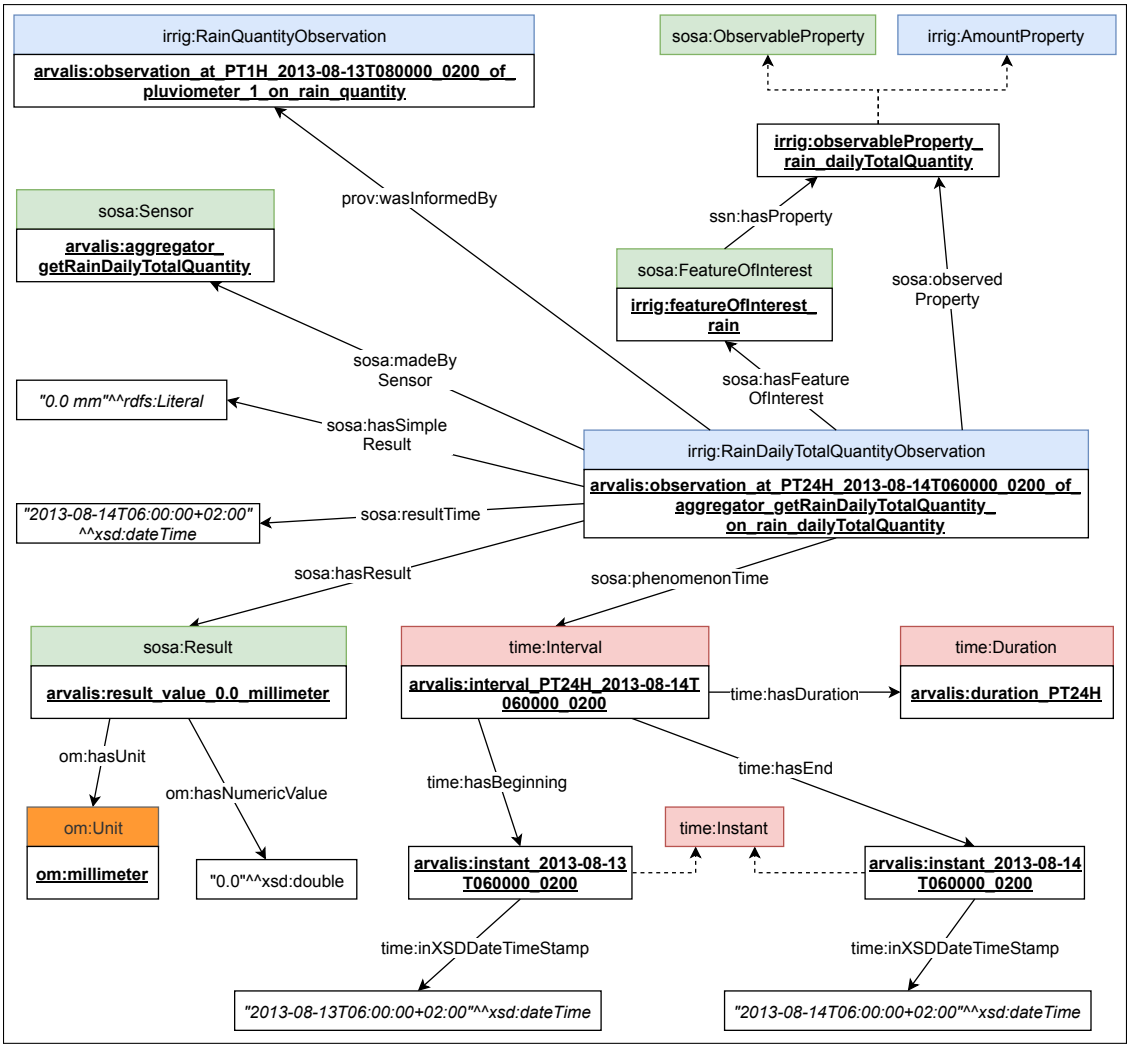



**Figure 32**: RainDailyTotalQuantity observation on 14/08/2013

information about the hour, minute, and second of the computation. For synchronization purposes, this precise time is fixed to the time of the deduction computation `[d+1 06:00:00]`. In Figure 32, this precise time is at 06:00:00 on 15/08/2013.

- To represent the fact that an observation concerns the data collected on the day d, the object property `sosa:phenomenonTime` is used. In Figure 32, the phenomenon time is represented by an instance of the class **time:Interval** for a period of `[14/08/2013 06:00:00, 15/08/2013 06:00:00[`, which is associated with two instances of the class **time:Instant**. The first one represents the beginning of the interval: `[14/08/2013 06:00:00]`. The second one represents the end of the interval: `[15/08/2013 06:00:00]`. Each

**time:Instant** instance uses a time:inXSDDateTimeStamp data property to store the associated date-time value. The `sosa:phenomenonTime` object property links the deduction individual to the interval individual.

- A computation of daily total rain quantity is based on several pluviometer measurements. To represent this fact, the instance of the class **irrig:RainDailyTotalQuantityObservation** is linked to several instances of the class **irrig:RainQuantityObservation** via the `prov:wasInformedBy` object property, as shown in Figure 32.
- The result of the computation is represented by an instance of the class **sosa:Result**. This instance is described by a unit and a value, as shown in Figure 32.

#### 3.4.3.3 DelayDurationObservation Modeling

The daily total rain quantity data can be processed to produce the delay duration. The delay duration is the number of days when irrigation should be postponed. To executing this aggregation process, the function `getDelay(d)` is run. Table 9 presents this function. This function returns the number of days based on the daily total rain quantity of the day d.

**Table 9**: Function getDelay(d)

$$\textbf{\textit{getDelay}}(d) = \begin{cases} \textbf{\textit{Quotient}}(\textbf{\textit{getRainDailyTotalQuantity}}(p,d), \mathbf{5}) & \\ \qquad \text{if getRainDailyTotalQuantity(p,d)} >= 10\text{mm} & \\ \mathbf{0} \qquad \text{if getRainDailyTotalQuantity(p,d)} < 10\text{mm} & \end{cases}$$

where

- d: a specific day, e.g., 14/08/2013
- getRainDailyTotalQuantity(p,d): returns the total of rain quantity measured by the pluviometer p during the interval [d 06:00:00, d+1 06:00:00[.
- Quotient: returns the integer part (or whole part) of a division operation.
- getDelay(d): is the duration of the delay period associated with the day d.

Figure 33 presents a sample of the DelayDuration observation on 14/08/2013 to be modeled using CASO and IRRIG.

- The instance of the class **irrig:DelayDurationObservation** represents the computation of the function `getDelay(d)`, as shown in Figure 33. By definition, this individual is linked to the individual

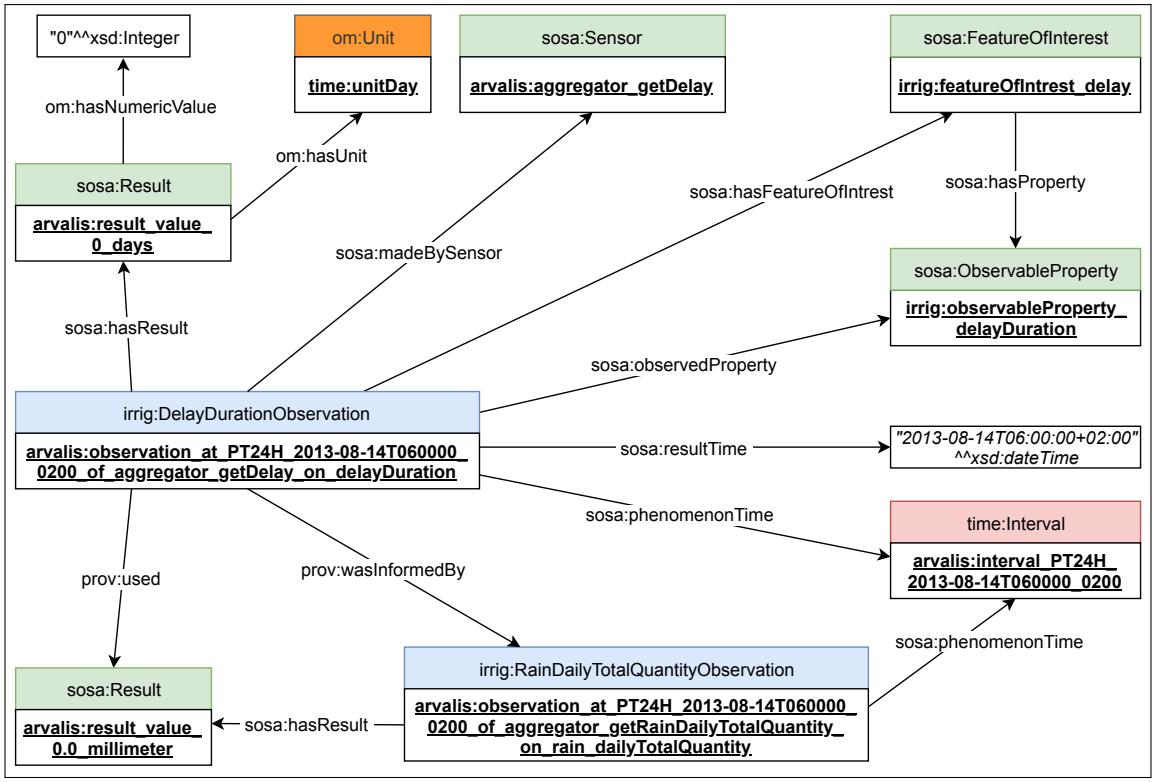



**Figure 33**: DelayDuration observation on 14/08/2013

irrig:observedProperty_delayDuration, an instance of the class **sosa:ObservableProperty**, and to the individual irrig:featureOfInterest_delay, an instance of the class **sosa:FeatureOfInterest**.

- The observation individual is linked to the actor that produces the result, that is, an aggregator runs the function `getDelay(d)`. Since there is no information about the aggregator that produces this computation in the Arvalis dataset, it is possible to represent it by the individual arvalis:aggregator_getDelay, an instance of the class **sosa:Sensor**. The observation individual is linked to the sensor individual via the `sosa:madeBySensor` object property, as shown in Figure 33.
- The aggregator produces the result once per day at a precise time. To present this time, the sosa:resultTime datatype property is used to link the observation individual to the `xsd:dateTime` value. In the Arvalis dataset, there is no information about the hour, minute, and second of the computation. For synchronization purposes, this precise time is fixed to the time of the deduction computation `[d+1 06:00:00]`. In Figure 33, this precise time is at 06:00:00 on 15/08/2013.

- To represent the fact that an observation concerns the data collected on the day d, the object property `sosa:phenomenonTime` is used. In Figure 33, the phenomenon time is represented by an instance of the class **time:Interval** for a period of `[14/08/2013 06:00:00, 15/08/2013 06:00:00[`, which is associated with two instances of the class **time:Instant**. The `sosa:phenomenonTime` object property links the deduction individual to the interval individual.
- The result of the computation is represented by an instance of the class **sosa:Result**. This instance is described by a unit and a value, as shown in Figure 33.
- A computation of delay duration is based on the daily total rain quantity data. To represent this fact, the instance of the class **irrig:DelayDuration** is linked to an instance of the class **irrig:RainDailyTotalQuantityObservation** via the `prov:wasInformedBy` object property, as shown in Figure 33. The observation individual is linked to the result individual via the `prov:used` object property.

#### 3.4.3.4 RainIntensityDeduction Modeling

The daily total rain quantity data are processed to deduct the daily state of the intensity for the rain. An inference engine runs this deduction process based on rules. The inference engine is SWRLAPI Drools Engine in the program Ontogen.

The rain intensity states are coded as states of the property RainIntensity. The list of RainIntensity states is as follows:

- **RainIntensity.Init**: This state represents the fact that the function `getRainDailyTotalQuantity(p,d)` will be evaluated from news measurements of one pluviometer. The pluviometer will perform several measurements per day. This state is not an output of the automata. It is a transitional state before determining the new state of the property RainIntensity.
- **RainIntensity.Light**: The amount of rain that falls is insignificant. It is insufficient to have any impact on irrigation.
- **RainIntensity.Moderate**: The amount of rain that falls is quite significant. The crops have no risk of water stress, so that irrigation may be postponed.
- **RainIntensity.Heavy**: The amount of rain that falls is highly significant. It can replace irrigation. Then, the crops have enough water after this rain.

The states of the property RainIntensity follow the order: `Light` < `Moderate` < `Heavy`. Certain thresholds are used to determine the states of the property RootZoneMoistureLevel from the result of the function

`getRainDailyTotalQuantity(p,d)`. Note that these thresholds depend on the variety of maize crops and soil types. There are four thresholds:

- Th_RQ_Minimum: This threshold is the minimum value that the function `getRainDailyTotalQuantity(p,d)` may return. It is 0 mm.
- Th_RQ_Low: This threshold is used to distinguish a light rain with a moderate rain. This threshold value in Arvalis experimentation is equal to 10 mm *(Arvalis et al., 2007)*.
- Th_RQ_High: This threshold is used to distinguish a heavy rain with a moderate rain. This threshold value in Arvalis experimentation is 40 mm *(Arvalis et al., 2007)*.
- Th_RQ_Maximum: This threshold is the maximum value that the function `getRainDailyTotalQuantity(p,d)` may return. It is fixed to 601 mm that even higher than the precipitation record in France [7].

The relations between thresholds and states of the property RainIntensity are presented in Figure 34. The blue squares represent the state of the property RainIntensity. The green squares represent the rain quantity threshold. The dashed line squares represent a value returned from the function `getRainDailyTotalQuantity(p,d)`. Each of the dashed line square's position relates to an upper threshold and a lower threshold. For example, if the value is superior or equal to Th_RQ_low (10) and inferior to the Th_RQ_high (40), then RainIntensity state is `Moderate`.

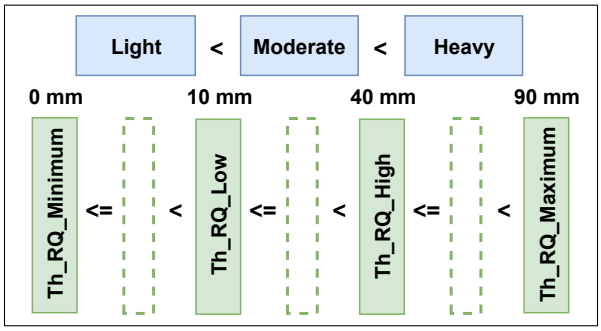



**Figure 34**: Thresholds of the states of the property RainIntensity

Figure 35 describes the state diagram that presents the inference conception to deduce the state of the property RainIntensity based on the result of the function `getRainDailyTotalQuantity(p,d)`. In this diagram, there are six states and six transitions. The six states include two default states of the UML state diagram (initial and final states) and four states defined for RainIntensity. The variable $t_{rain}$

[7] http://pluiesextremes.meteo.fr/france-metropole/Records-mondiaux.html

represents a clock dedicated to the property RainIntensity that has a time step of one day. Moreover, each state of RainIntensity in the diagram may contain internal transitions.

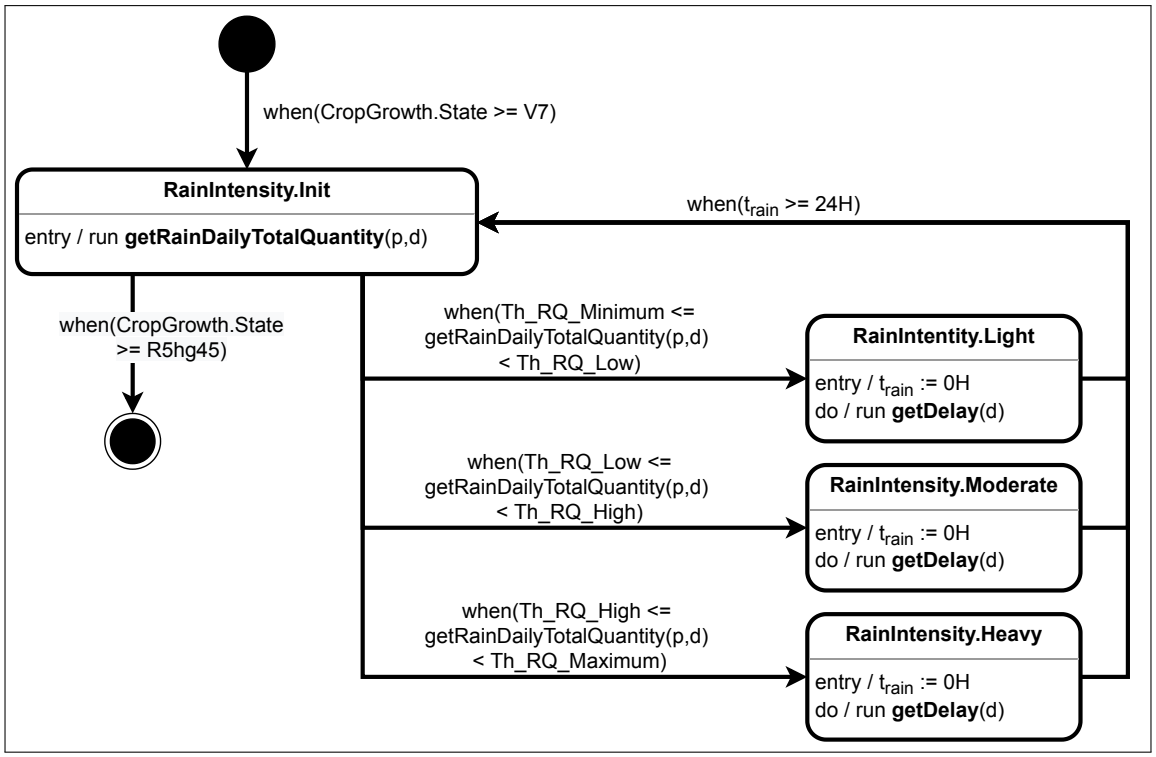



**Figure 35**: Automata of RainIntensity states and their transitions

- The transition from the initial state towards the state `Init` indicates when the state of the CropGrowth property is `V7` or any upper state; then, the property RainIntensity reaches the state `Init`. After reaching this state, a new evaluation of the function `getRainDailyTotalQuantity(p,d)` is performed based on daily pluviometer measurements.
- The transition from the state `Init` towards the state `Light` indicates that when the value of the function `getRainDailyTotalQuantity(p,d)` is inferior to Th_RQ_Low and superior or equal to Th_RQ_Minimum, then property RainIntensity reaches the state `Light`. After reaching this state, the function `getDelay(d)` is evaluated. Also, the $t_{rain}$ is reset.
- The transition from the state `Init` towards the state `Moderate` indicates when the value of the function `getRainDailyTotalQuantity(p,d)` is inferior to Th_RQ_High and superior or equal to Th_RQ_Low, then property RainIntensity reaches the state `Moderate`. After reaching this state, the function `getDelay(d)` is evaluated. Also, the $t_{rain}$ is reset.

- The transition from the state `Init` towards the state `Heavy` indicates when the value of the function `getRainDailyTotalQuantity(p,d)` is inferior to Th_RQ_Maximum and superior or equal to Th_RQ_High, the property RainIntensity reaches the state `Heavy`. After reaching this state, the function `getDelay(d)` is evaluated. Also, the $t_{rain}$ is reset.
- The transitions from one state among `Light`, `Moderate`, `Heavy` towards the state `Init` indicates a state change is possible only after 24 hours (($t_{rain}$ >= 24 h)?); in other words, a state is valid for a whole day. To take a new state change decisions, new daily evaluation of the function `getRainDailyTotalQuantity(p,d)` is performed.
- The transitions from one state among `Light`, `Moderate`, `Heavy` towards the final state indicate when the CropGrowth property reaches the state `R5hg45`, pluviometer measurements are terminated.

Figure 36 and Figure 37 present a sample of the RainIntensity deduction on 14/08/2013 to be modeled using CASO and IRRIG.

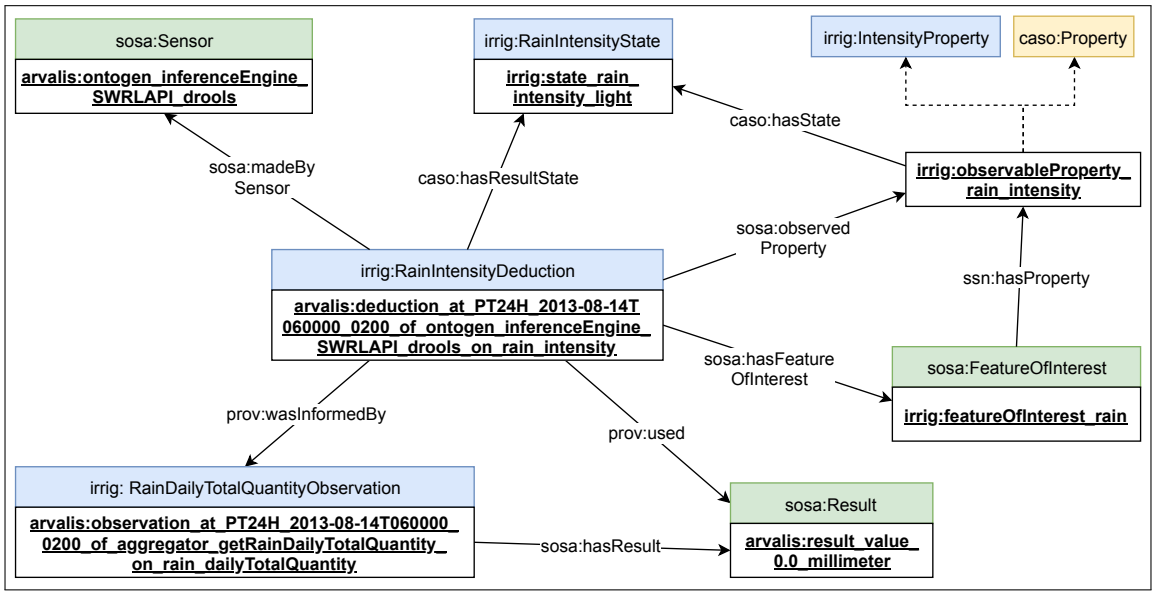



**Figure 36**: RainIntensity deduction on 14/08/2013 (part 1)

- An instance of the class **irrig:RainIntensityDeduction** represents a deduction process for the property RainIntensity, as shown in Figure 36. By definition, this individual is linked to the individual irrig:observedProperty_rain_intensity, an instance of **irrig:IntensityProperty**, and to the irrig:featureOfInterest_rain individual.
- The deduction individual is linked to an actor that produces the result. The

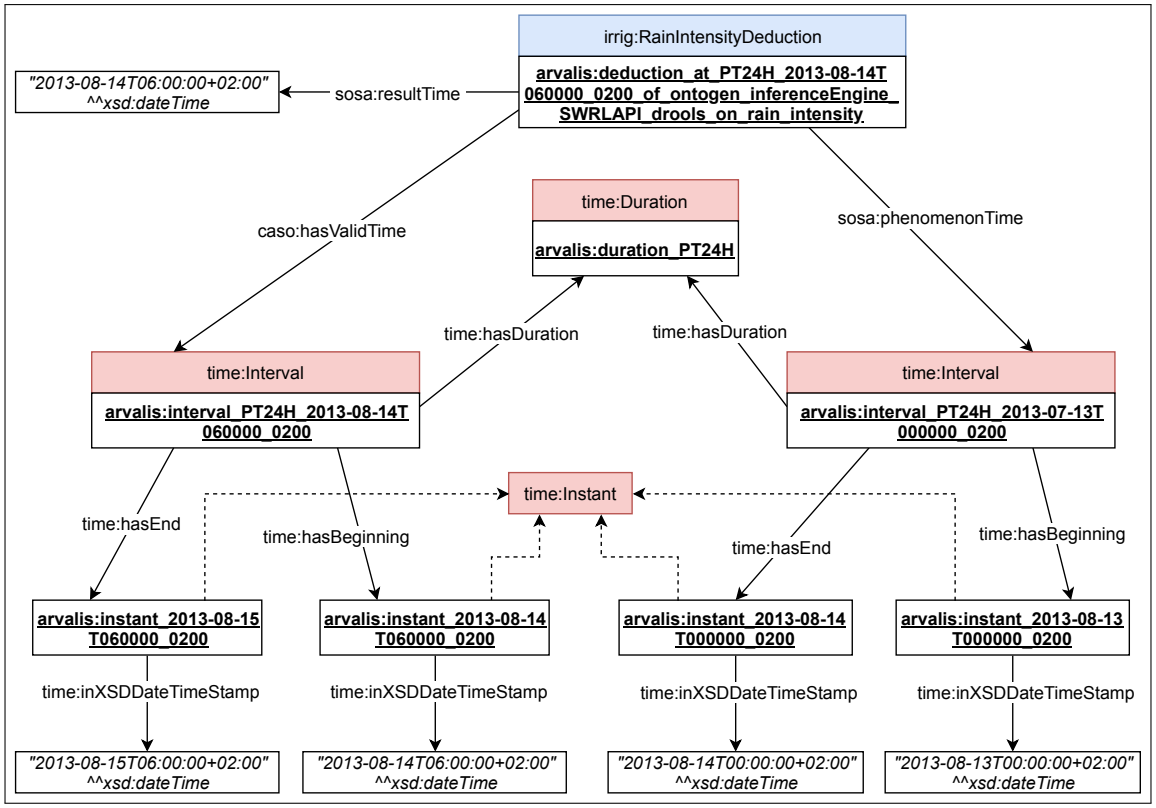



**Figure 37**: RainIntensity deduction on 14/08/2013 (part 2)

actor is an SWRLAPI Drools Engine that is represented by the individual arvalis:ontogen_inferenceEngine_SWRLAPI_drools, an instance of the class **sosa:Sensor**. The deduction individual is linked to the actor via the `sosa:madeBySensor` object property, as shown in Figure 36.

- A RainIntensity deduction is based on an observation individual that represents a computation of the function `getRainDailyTotalQuantity(p,d)`. Thus, the deduction individual in Figure 36 is based on the observation individual presented in Figure 32. The deduction individual is linked to the instance of the class **irrig:RainDailyTotalQuantityObservation** via the `prov:wasInformedBy` object property.
- The result of a RainIntensity deduction is an instance of the class **irrig:RainIntensityState**. The deduction individual is linked to its state result by the `caso:hasResultState` object property, as shown in Figure 36. The possible states are shown in Figure 38.
- To store the time of the deduction, the sosa:resultTime datatype property is used to link the deduction individual to the xsd:dateTime value corresponding to the result time at `[d+1 06:00:00]`. In Figure 37, the result time is at 06:00:00 on

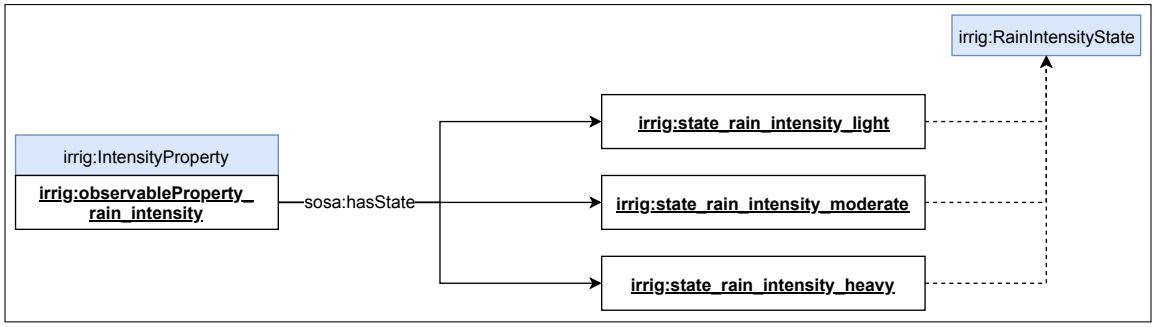



**Figure 38**: States of the property RainIntensity

15/08/2013.

- To represent the fact that a deduction concerns the data collected on the day d, the object property `sosa:phenomenonTime` is used. In Figure 37, the day d is on 14/08/2013. It is represented by an instance of the class **time:Interval**, which is described by two instances of the class **time:Instant**. The first one represents the beginning of the interval: `[14/08/2013 00:00:00]`. The second one represents the end of the interval: `[15/08/2013 00:00:00]`. The `sosa:phenomenonTime` object property links the deduction individual to the interval individual.
- The deduction result for day d is valid for 24 h: `[d+1 06:00:00, d+2 06:00:00[`, that is, no new deduction will be performed during this valid time. In Figure 37, the valid time is represented by an instance of the class **time:Interval** for a period of `[15/08/2013 06:00:00, 16/08/2013 06:00:00[`, which is associated with two instances of the class **time:Instant** as presented above. The `caso:hasValidTime` object property links the deduction individual to the valid time.

Figure 39 supports to clarify the synchronization of all the computations of the rain intensity workflow.

#### 3.4.3.5 RainIntensityDeduction Reasoning

Figure 35 gives a global view of the mechanism to deduct a RainIntensity state. In reasoning, one rule can represent this mechanism. This rule is coded in SWRL. Table 10 contains necessary information of the rule. The code in the box **Rule in SWRL** presents the logic of the reasoning process. Detail of the code is as follows.

- The first paragraph is one premise. It means to find a rain intensity deduction of a day.
- The second paragraph is one premise. It means to get the data value of the daily rain quantity observation of a day.

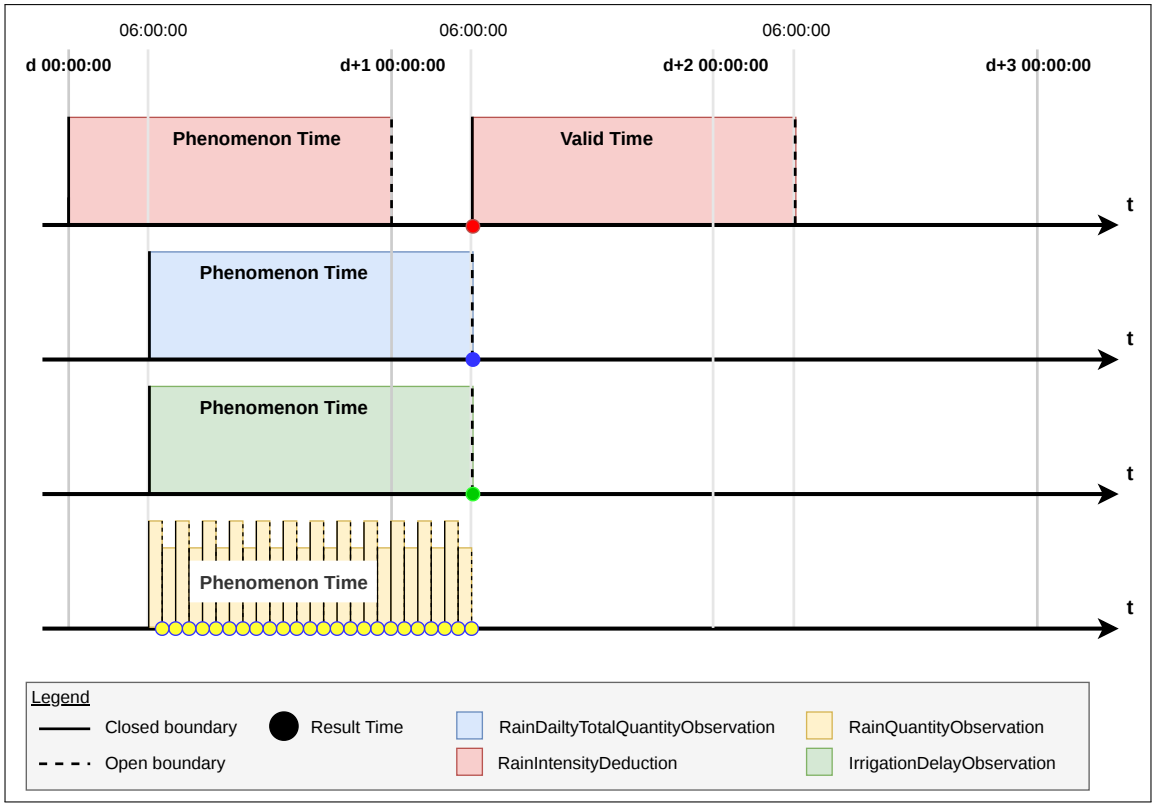



**Figure 39**: Time scales of the rain intensity workflow

- The third paragraph is one premise. It means getting the data value of the upper boundary and the lower boundary of every state of rain intensity.
- The fourth paragraph is one premise. It means comparing this data value of the daily rain quantity observation of a day to check if it is inferior to the value of the upper boundary and superior or equal to the lower boundary of a state of rain intensity.
- The final paragraph is one deduction. It means a rain intensity deduction has the state satisfy the condition in the fourth paragraph.

### 3.4.4 Soil Moisture Workflow Data Modeling and Reasoning

The IRRINOV® method evaluates the water needs of crops by estimating the root zone moisture. Root zone moisture is the quantity of "water remaining in the depth of soil accessed by a plant" *(International Atomic Energy Agency (IAEA), 2008)*. It depends on the type of soil, the depth of crop root, and the irrigation method. Its goal is to evaluate the amount of irrigation water required by the plants *(Brouwer et al., 1989)*. The system uses six tensiometers to evaluate the root zone moisture.

**Table 10**: Rule for the property RainIntensity to reach a state

| **Code:** | **Full name:** |
|---|---|
| ADv122019-RI | ArvalisData-IrrigVersion122019-RainIntensity |
| **Description:**<br>The goal of this rule is to determine the state of RainIntensity. The rule implements the transition from the Init state to one of the state Light, Moderate and Heavy. The input of this rule is the rain daily total quantity (?result_observation_rain_total_quantity). The output of this rule is the Rain Intensity deduction (?deduction_rain_intensity). The mechanism of this rule is to check if the value of the rain daily total quantity is in the value domain of two thresholds; it then concludes the state correspondingly. | |
| **Rule in SWRL:**<br>irrig:RainIntensityDeduction(?deduction_rain_intensity) ^<br><br>prov:used(?deduction_rain_intensity, ?result_observation_rain_total_quantity) ^<br>om:hasNumericalValue(?result_observation_rain_total_quantity, ?value_result_observation) ^<br><br>caso:hasState(irrig:observableProperty_rain_intensity, ?rain_intensity_state) ^<br>caso:hasOpenUpperBoundary(?rain_intensity_state, ?boundary_upper) ^<br>caso:boundaryValue(?boundary_upper, ?value_boundary_upper) ^<br>caso:hasClosedLowerBoundary(?rain_intensity_state, ?boundary_lower) ^<br>caso:boundaryValue(?boundary_lower, ?value_boundary_lower) ^<br><br>swrlb:lessThan(?value_result_observation, ?value_boundary_upper) ^<br>swrlb:greaterThanOrEqual(?value_result_observation, ?value_boundary_lower)<br><br>-> caso:hasResultState(?deduction_rain_intensity, ?rain_intensity_state) | |

In the soil moisture workflow, data related to soil moisture is processed through several processes. The input of this workflow is the soil moisture at a different specific depth in the soil. The output of this workflow is the state of the property RootZoneMoistureLevel. As in Figure 23, the soil moisture workflow composes of six following processes:

- **Soil30cmDepthMoistureObservation**: an observation that provides the moisture of the feature of interest representing the soil layer at a 30 cm depth (Soil30cmDepth).
- **Soil60cmDepthMoistureObservation**: an observation that provides the moisture of the feature of interest representing the soil layer at a 60 cm depth (Soil60cmDepth).
- **Soil30cmDepthDailyAverageMoistureObservation**: an aggregation that provides the daily average moisture of the feature of interest Soil30cmDepth. It

equals the daily average of the property Soil30cmDepthMoisture.

- **Soil60cmDepthDailyAverageMoistureObservation**: an aggregation that provides the daily average moisture of the feature of interest Soil60cmDepth. It equals the daily average of the property Soil60cmDepthMoisture.
- **RootZoneDailyAverageMoistureObservation**: an aggregation that provides the daily average moisture of the feature of interest RootZone. It equals a spatial aggregation of the properties Soil30cmDepthDailyAverageMoisture and Soil60cmDepthDailyAverageMoisture.
- **RootZoneDailyAverageMoistureDeduction**: a deduction that provides the state of the property RootZoneMoistureLevel.

#### 3.4.4.1 Soi30cmDepthMoistureObservation Modeling

A tensiometer measures the soil tension *(Migliaccio et al., 2002)*. The soil tension affects the water extraction capability of crops as the following principles:

- When the soil tension is low, the crops require less energy to extract soil water and the soil moisture level is high.
- When the soil tension is high, the crops have to use a large amount of energy to extract soil water and the soil moisture level is low.

The soil tensiometers used in the IRRINOV® method are Watermark probes. Watermark probe p measures the soil moisture at time t of day d. The unit of the Watermark probe is the cbar. Note that a measurement value of 199 cbar indicates an error in the measurement process. Thus, the cleaning process is applied to remove all 199 cbar values. To executing this process, the function `getValidMoistureProbe(p,t)` is run. This function returns only the valid moisture value measured by probe p at time t. If the moisture value is not valid, the function returns nothing.

Figure 40 presents a sample of the Soil30cmDepthMoisture observation at 08:00:00 on 14/08/2013 to be modeled using CASO and IRRIG.

- An instance of the class **irrig:Soil30cmDepthMoistureObservation** represents the computation of the function `getValidMoistureProbe(p,t)`. It is an observation of a moisture property related to the soil layer at a 30 cm depth, as shown in Figure 40. By definition, instances of **irrig:Soil30cmDepthMoistureObservation** are linked to the individual irrig:observedProperty_soil30cmDepth_moisture, an instance of the class **irrig:MoistureProperty**, and to the individual irrig:featureOfInterest_soil30cmDepth, an instance of the class **sosa:FeatureOfInterest**.

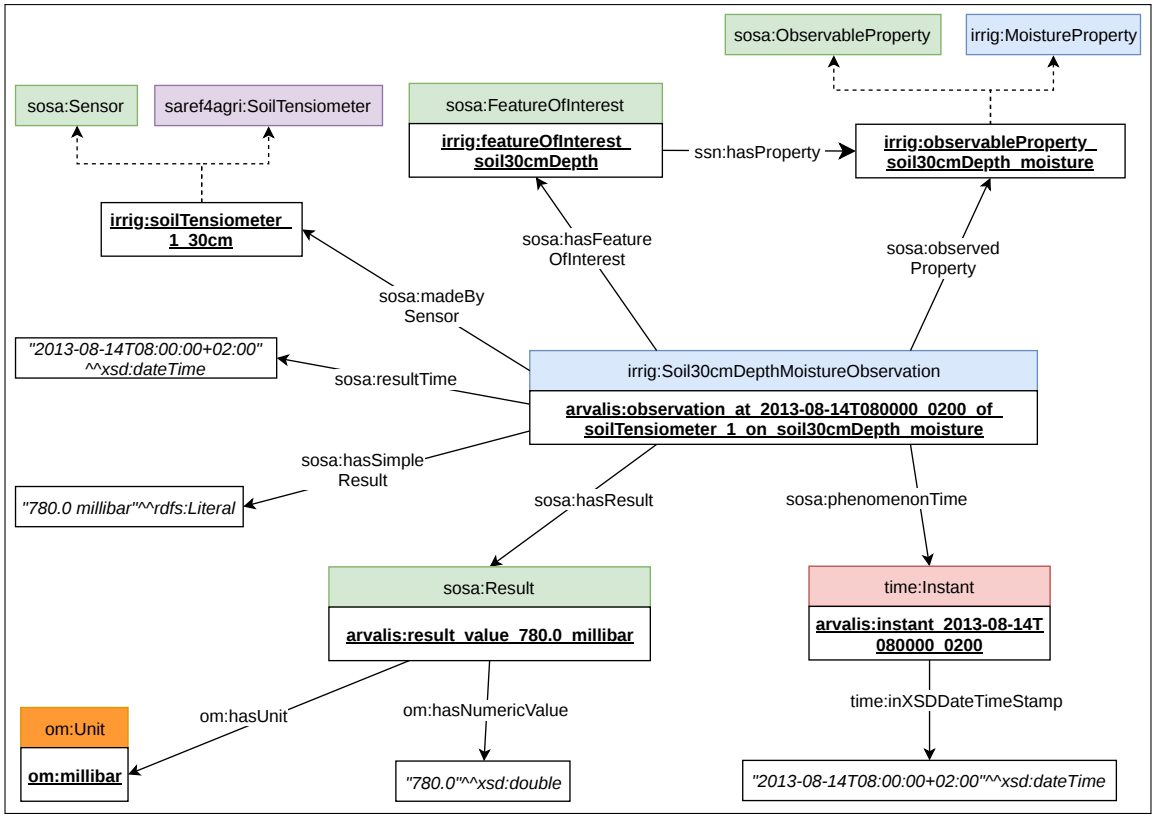



**Figure 40**: Soil30cmDepthMoisture observation at 08:00:00 on 14/08/2013

- The observation individual is linked to the sensor that made the observation. Assuming that the node, which contains a tensiometer probe, also run the cleaning process. A tensiometer is represented by an instance of the class **sosa:Sensor**, is also an instance of the class **saref4agri:SoilTensiometer**. Arvalis identifies their tensiometers by a number and depth. This information is reused to create a name for each instance of **saref4agri:SoilTensiometer**. For example, Figure 40 contains the individual arvalis:soilTensiometer_1_30cm. The `sosa:madeBySensor` object property links the observation individual to the sensor individual.
- A tensiometer measurement is an instantaneous observation. Note that the measurements of tensiometers are unavailable in the Arvalis dataset. However, it is possible to imagine that tensiometer measurements could happen at any time of the day. Figure 40 presents a measurement that happens at 08:00:00 on 14/08/2013. To store the time information of the measurement, this model uses an instance of the class **time:Instant**. The time:inXSDDateTimeStamp data property stores the associated date-time value. The `sosa:phenomenonTime` object property links the observation individual to the instant individual.

- To present the time when the tensiometer generates the measurement, the sosa:resultTime datatype property is used to link the observation individual to the `xsd:dateTime` value corresponding to a precise time. In Figure 40, this precise time is at 08:00:00 on 14/08/2013.
- The result of the tensiometer measurement is presented by an instance of the class **sosa:Result**. A value and a unit are required to describe this result. The instance of **sosa:Result** is linked to the data value via the om:hasNumericValue datatype property. The instance of **sosa:Result** is linked to the individual om:millibar, an instance of the class **om:Unit**, via the `om:hasUnit` object property. Note that in the Arvalis dataset, the unit of tensiometer measurement is the cbar. However, this unit has not yet been defined in any ontology. Thus, this model uses the millibar (mbar) as the unit for moisture, as shown in Figure 40. This unit is defined in some well-known ontologies, such as OM.

#### 3.4.4.2 Soil30cmDepthDailyAverageMoistureObservation Modeling

The data recorded in the Arvalis dataset are not the measurements of tensiometers, but the aggregations of those measurements: the average soil moisture on a day. The measured data collected from each tensiometer are processed to become the daily aggregated soil moisture. To executing this aggregation process, the function `getDailyAggregatedMoisture(p,d)` is run. Table 11 presents the function `getDailyAggregatedMoisture(p,d)`. This function returns the average value of all the valid moisture values measured by probe p during the interval `[d 00:00:00, d+1 00:00:00[`.

**Table 11**: Function getDailyAggregatedMoisture(p,d)

$$\boldsymbol{getDailyAggregatedMoisture(p,d)} = \frac{\sum_{i=1}^{n} \boldsymbol{getValidMoistureProbe(p,t_i)}}{\boldsymbol{n}}$$

on the condition that $t_i \subset$ [d 00:00:00, d+1 00:00:00[

where

- d: a specific day, e.g., 14/08/2013
- $t_i$: a specific instant of day d, e.g., 08:00:00 on 14/08/2013
- getValidMoistureProbe(p, $t_i$): the valid moisture value measured by the probe p at the specific time $t_i$.
- n: the number of valid moisture values measured during day d.

In this model, the aggregation process is represented by observation of each

tensiometer on the daily average soil moisture at different depths. In detail, three soil tensiometers observe the moisture at a 30 cm depth, and three others observe the moisture at a 60 cm depth. In the Arvalis dataset, each observation was stored as two data points: (1) the daily aggregated soil moisture in cbar, and (2) the date of observation (year, month, day).

Figure 41 presents a sample of the Soil30cmDepthDailyAverageMoisture observation on 14/08/2013 to be modeled using CASO and IRRIG.

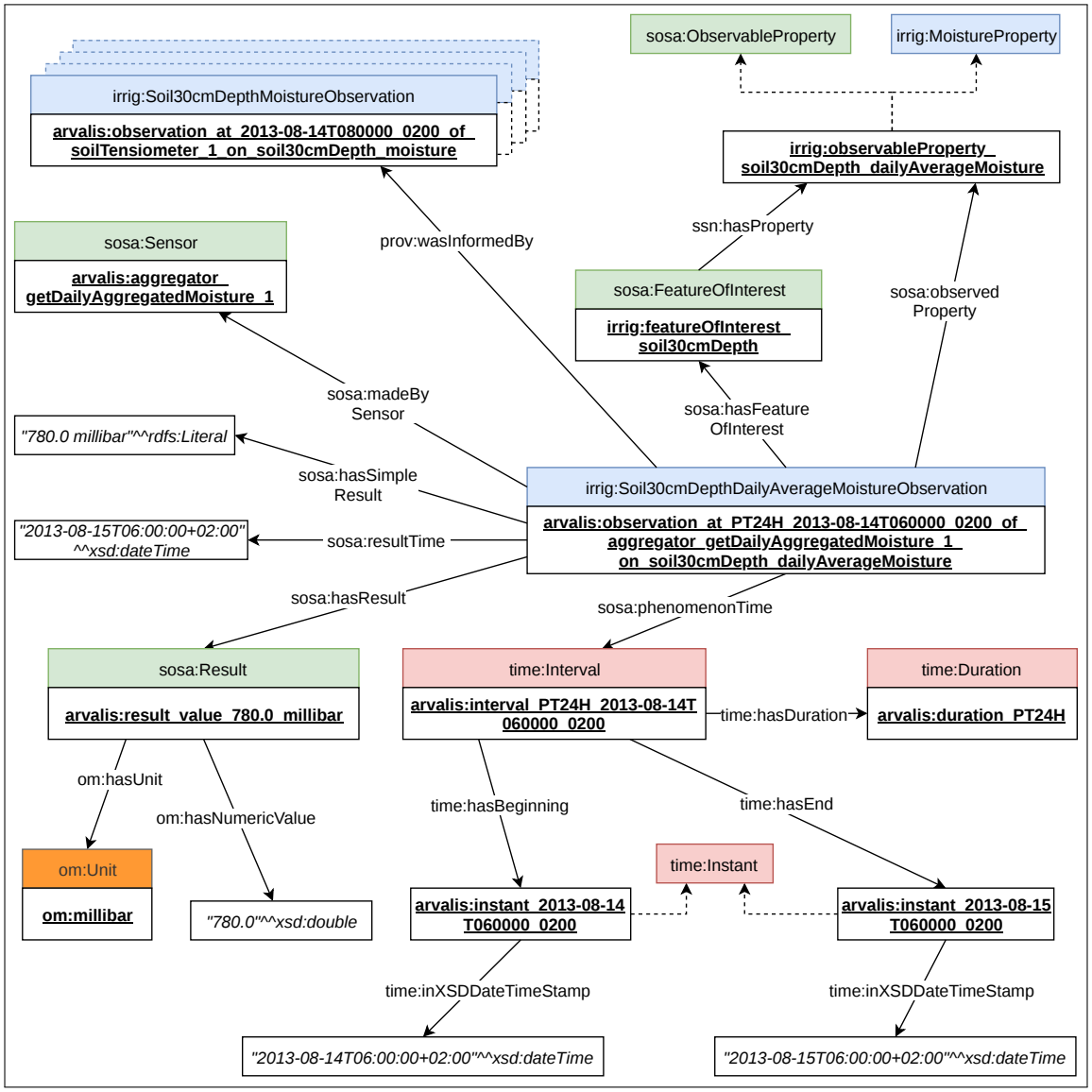



**Figure 41**: Soil30cmDepthDailyAverageMoisture observation on 14/08/2013

- The instance of the class **irrig:Soil30cmDepthDailyAverageMoistureObservation** represents the computation of the daily average of all measurements provided

by one tensiometer, as shown in Figure 41. By definition, instances of **irrig:Soil30cmDepthDailyAverageMoistureObservation** are linked to the individual irrig:observedProperty_soil30cmDepth_dailyAverageMoisture, an instance of the class **irrig:MoistureProperty**, and to the individual irrig:featureOfInterest_soil30cmDepth.

- The observation individual is linked to the actor that produces the result, that is, a software agent that run the function `getDailyAggregatedMoisture(p,d)`. Since there is no information on the software agent that produces this computation in the Arvalis dataset, it is possible to represent it by an instance of the class **sosa:Sensor**. To avoid errors, a software agent for each tensiometer is created based on the tensiometer name. For example, the sensor individual for the first tensiometer is arvalis:aggregator_getDailyAggregatedMoisture_1, as shown in Figure 41. The observation individual is linked to the software agent via the `sosa:madeBySensor` object property
- The software agent produces the result once per day at a precise time. To present this time, the sosa:resultTime datatype property is used to link the observation individual to the `xsd:dateTime` value. In the Arvalis dataset, there is no information about the hour, minute, and second of the computation. For synchronization purposes, this precise time is fixed to the time of the deduction computation `[d+1 06:00:00]`. In Figure 41, this precise time is at 06:00:00 on 15/08/2013.
- The computation calculates the daily average. Figure 41 presents the average on 14/08/2013 as observation during a day of 24 hours. This interval is represented by an instance of the class **time:Interval**, which is described by two instances of **time:Instant**. The first one represents the beginning of the interval: `[14/08/2013 06:00:00]`. The second one represents the end of the interval: `[15/08/2013 06:00:00]`.
- A computation of daily average is based on several tensiometer measurements. To represent this fact, the instance of the class **irrig:Soil30cmDepthDailyAverageMoistureObservation** is linked to several instances of the class **irrig:Soil30cmDepthMoistureObservation** via the `prov:wasInformedBy` object property, as shown in Figure 41.
- The result of the computation is represented by an instance of the class **sosa:Result**. This instance is described by a unit and a value, as shown in Figure 41.

#### 3.4.4.3 RootZoneDailyAverageMoistureObservation Modeling

The six daily aggregated soil moisture data corresponding to six tensiometers are processed to become the daily aggregated root zone moisture. To executing this aggregation process, the function `getRootZoneDailyAverageMoisture(d)` is run. Table 12 presents this function. This function returns the average value of all the valid moisture values measured by probe p during the interval of 24 hours. Suppose that p1, p2, and p3 are the three probes at a 30 cm depth, and p4, p5, and p6 are the three probes at a 60 cm depth. The function getRootZoneDailyAverageMoisture(d) returns the sum of the two median values: (1) the median of the getDailyAggregatedMoisture(p,d) values of three probes at a 30 cm depth, and (2) the median of the getDailyAggregatedMoisture(p,d) values of three probes at a 60 cm depth.

**Table 12**: Function getRootZoneDailyAverageMoisture(d)

$$
\begin{aligned}
&\boldsymbol{getRootZoneDailyAverageMoisture(d)} = \\
&\qquad \boldsymbol{Median(getDailyAggregatedMoisture(p1,d),} \\
&\qquad\qquad \boldsymbol{getDailyAggregatedMoisture(p2,d),} \\
&\qquad\qquad \boldsymbol{getDailyAggregatedMoisture(p3,d))} \\
&\quad + \boldsymbol{Median(getDailyAggregatedMoisture(p4,d),} \\
&\qquad\qquad \boldsymbol{getDailyAggregatedMoisture(p5,d),} \\
&\qquad\qquad \boldsymbol{getDailyAggregatedMoisture(p6,d))}
\end{aligned}
$$

where

- p1, p2, and p3: three watermark probes placed at a 30 cm depth
- p4, p5, and p6: three watermark probes placed at a 60 cm depth
- getDailyAggregatedMoisture($p_i$,d): returns the average moisture value measured by the probe $p_i$ during the interval [d 00:00:00, d+1 00:00:00[ (see Table 11)
- Median(x, y, z): is the median value of x, y and z.

Figure 42 presents a sample of the RootZoneDailyAverageMoisture observation on 14/08/2013 to be modeled using CASO and IRRIG.

- An instance of the class **irrig:RootZoneDailyAverageMoistureObservation** represents the computation of the daily average moisture for root zone. The computation of the daily average is presented in Table 12. By definition, instances of **irrig:RootZoneDailyAverageMoistureObservation** are linked to the individual irrig:observedProperty_rootZone_dailyAverageMoisture, an instance of the class **irrig:MoistureProperty**, and to the individual

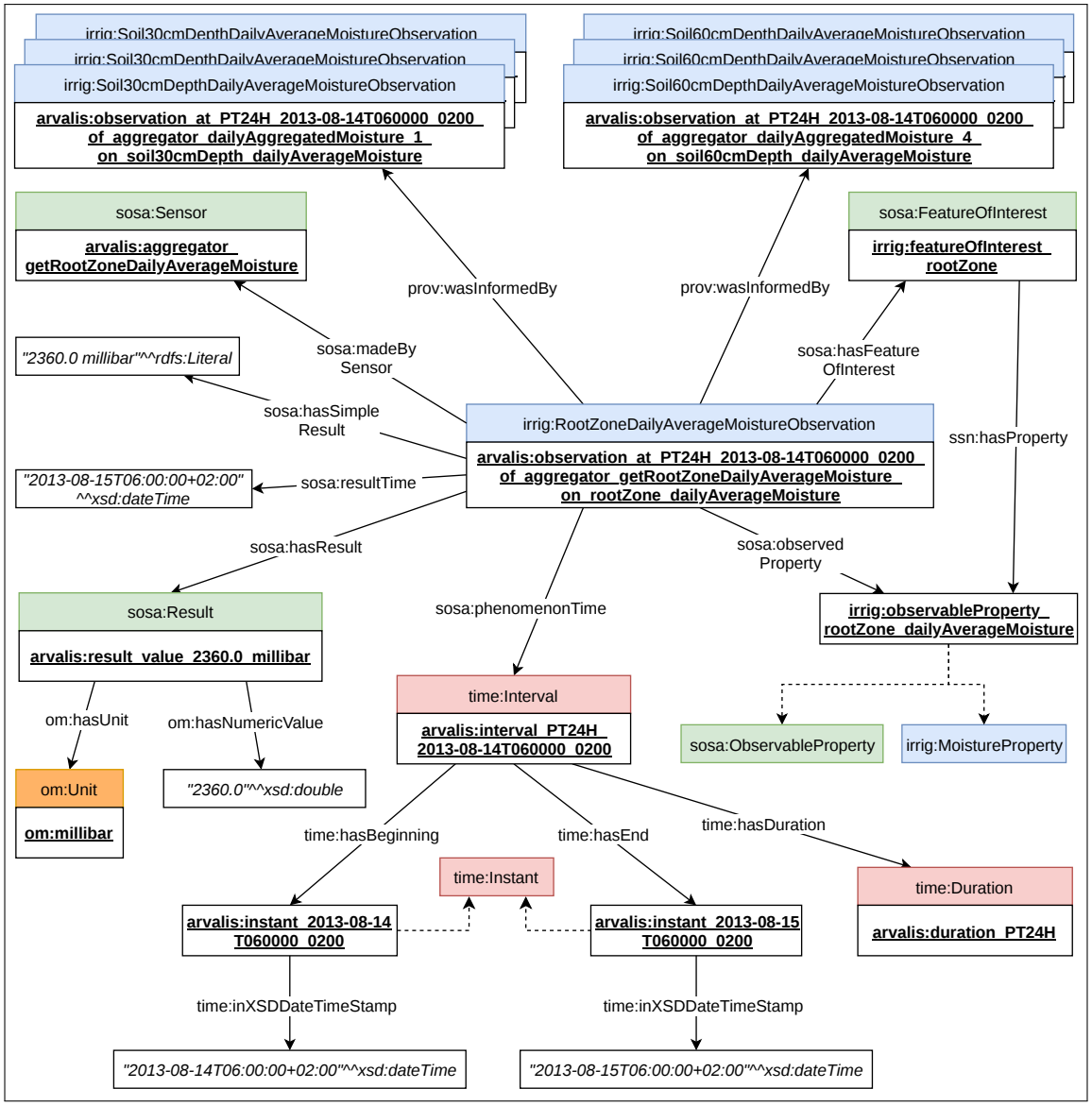



**Figure 42**: RootZoneDailyAverageMoisture observation on 14/08/2013

irrig:featureOfInterest_rootZone, an instance of **sosa:FeatureOfInterest**, as shown in Figure 42.

- The observation individual is linked to the sensor that produces the result, that is, a software that computes the equation presented in Table 12. The software program is named arvalis:aggregator_getRootZoneDailyAverageMoisture. This individual is an instance of the class **sosa:Sensor**. The observation individual is linked to the sensor via the `sosa:madeBySensor` object property, as shown in Figure 42.

- The computation calculates the daily average root zone moisture, that is, average moisture of root zone observed for 24 hours. Figure 42 presents this aggregated value on 14/08/2013 as observation during a day of 24 hours. This interval is represented by an instance of the class **time:Interval**, which is described by two instances of **time:Instant**. The first one represents the beginning of the interval: `[14/08/2013 06:00:00]`. The second one represents the end of the interval: `[15/08/2013 06:00:00]`.
- The software program produces the result once per day at a precise time. To present this time, the sosa:resultTime datatype property is used to link the observation individual to the `xsd:dateTime` value. In the Arvalis dataset, there is no information about the hour, minute, and second of the computation. For synchronization purposes, this precise time is fixed to the time of the deduction computation `[d+1 06:00:00]`. In Figure 42, this precise time is at 06:00:00 on 15/08/2013.
- A computation of root zone daily average is based on the daily aggregated data corresponding to six tensiometers. To represent this fact, the instance of the class **irrig:RootZoneDailyAverageMoistureObservation** is linked to three instances of the classes **irrig:Soil30cmDepthMoistureObservation** and three instances of the classes **irrig:Soil60cmDepthMoistureObservation** via the `prov:wasInformedBy` object property, as shown in Figure 42.
- The result of the computation is presented by an instance of the class **sosa:Result**. This instance is described by a unit and a value, as shown in Figure 42.

#### 3.4.4.4 RootZoneMoistureLevelDeduction Modeling

The daily aggregated root zone moisture data are processed to deduct the daily state of the moisture level for the root zone. An inference engine runs this deduction process based on rules. The inference engine is SWRLAPI Drools Engine in the program Ontogen.

The root zone moisture level states are coded as states of the property RootZoneMoistureLevel. The list of RootZoneMoistureLevel states is as follows:

- **RootZoneMoistureLevel.Init**: The state represents the fact that the function `getRootZoneDailyAverageMoisture(d)` will be evaluated from new measurements of tensiometers. The tensiometers will perform several measurements per day. This state is not an output of the automata. It is a transitional state before determining the new state of the RootZoneMoistureLevel property.

- **RootZoneMoistureLevel.Saturated**: The root zone is saturated by water. Crops do not need extra water. Any machine is allowed to access the plot.
- **RootZoneMoistureLevel.VeryHigh**: The root zone contains a significantly high amount of water. Crops do not need water regardless of its growth stage.
- **RootZoneMoistureLevel.High**: The root zone contains a high amount of water. Crops need water at specific growth stages.
- **RootZoneMoistureLevel.Average**: The root zone contains an average amount of water. Crops need water at specific growth stages.
- **RootZoneMoistureLevel.Low**: The root zone contains a low amount of water. Crops need water at specific growth stages.
- **RootZoneMoistureLevel.VeryLow**: The root zone contains an insignificant amount of water. Crops need water at specific growth stages.
- **RootZoneMoistureLevel.Dry**: The root zone is dry. Crops need water regardless of its growth stage.

The states of the property RootZoneMoistureLevel follow the order: `VeryLow` $<$ `Low` $<$ `Average` $<$ `High` $<$ `VeryHigh` $<$ `Saturated`. Those states follow the inverse order of the value of the function `getRootZoneDailyAverageMoisture(d)`. Certain thresholds are used to determine the state of the property RootZoneMoistureLevel from the result of the function `getRootZoneDailyAverageMoisture(d)`. Note that these thresholds depend on the variety of maize crops and soil types. There are eight thresholds:

- **Th_ST_Minimum**: The smallest value measured by a Watermark probe. In the Watermark's specification, it is fixed at 0 cbar.
- **Th_ST_Saturation**: This threshold value in the Arvalis dataset is 10 cbar.
- **Th_ST_VeryHigh**: This threshold value in the Arvalis dataset is 70 cbar.
- **Th_ST_High**: This threshold value in the Arvalis dataset is 120 cbar.
- **Th_ST_Average**: This threshold value in the Arvalis dataset is 140 cbar.
- **Th_ST_Low**: This threshold value in the Arvalis dataset is 150 cbar.
- **Th_ST_VeryLow**: This threshold value in the Arvalis dataset is 160 cbar.
- **Th_ST_Maximum**: This threshold value is set to 301 cbar, which is higher than any measurement value provided by any Watermark probe. The maximal measurement value that a Watermark probe can reach ranges from 0 to 200. The maximum value after multiplying with the maximum coefficient provided by Watermark's manufacturer is 1.5 (from 2002 until now). Then the maximal soil tension value is 300 cbar (200 cbar * 1.5).

The relations between thresholds and states of the property RootZoneMoistureLevel are presented in Figure 43. The blue squares represent the state of the property RootZoneMoistureLevel. The green squares represent the soil tension thresholds. The dashed line squares represent a value of the function `getRootZoneDailyAverageMoisture(d)`. Each of the squares' positions shows the upper threshold and the lower threshold. For example, if the value of the function `RootZoneDailyAverageMoisture(d)` is superior or equal to Th_ST_Low (150) and inferior to the Th_ST_VeryLow (160), then the RootZoneMoistureLevel state is `VeryLow`.

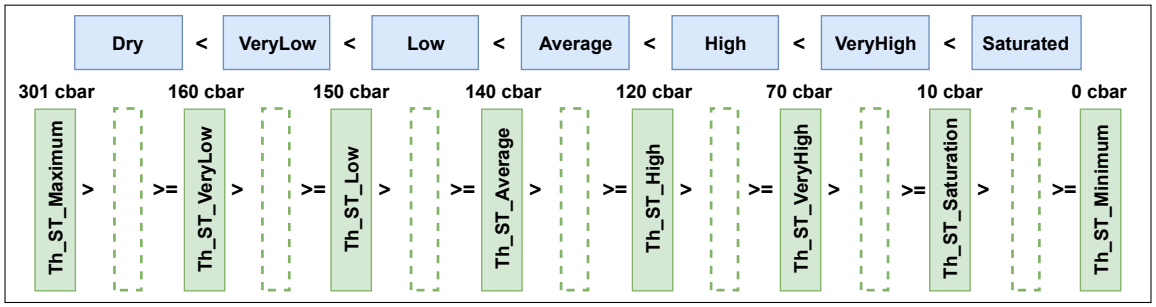



**Figure 43**: Thresholds of the states of the property RootZoneMoistureLevel

Figure 44 describes the state diagram that presents the inference conception to deduce the state of the RootZoneMoistureLevel property based on the result of the function `getRootZoneDailyAverageMoisture(d)`. In this diagram, there are ten states and ten transitions. The ten states include two default states of the UML state diagram (initial and final states) and eight states defined for RootZoneMoistureLevel. The variable $t_{soil}$ represents a clock dedicated to the RootZoneMoistureLevel property that has a time step of one day. Moreover, each state of RootZoneMoistureLevel in the diagram may contain internal transitions.

- The transition from the initial state towards the state `Init` indicates when the state of the CropGrowth property is `V7` or any upper state; then, the RootZoneMoistureLevel property reaches the state `Init`. After reaching this state, a new evaluation of the function `getRootZoneDailyAverageMoisture(d)` is performed based on daily tensiometer measurements.
- The transition from the state `Init` towards the state `Saturated` indicates that when the value of the function RootZoneDailyAverageMoisture(d) is equal or superior to Th_ST_Minimum and inferior to Th_ST_Saturation, then the RootZoneMoistureLevel property reaches the `Saturated` state. After reaching this state, the clock $t_{soil}$ is reset.
- The transition from the state `Init` towards the state `VeryHigh` indicates that when the value of the function getRootZoneDailyAverageMoisture(d) is equal

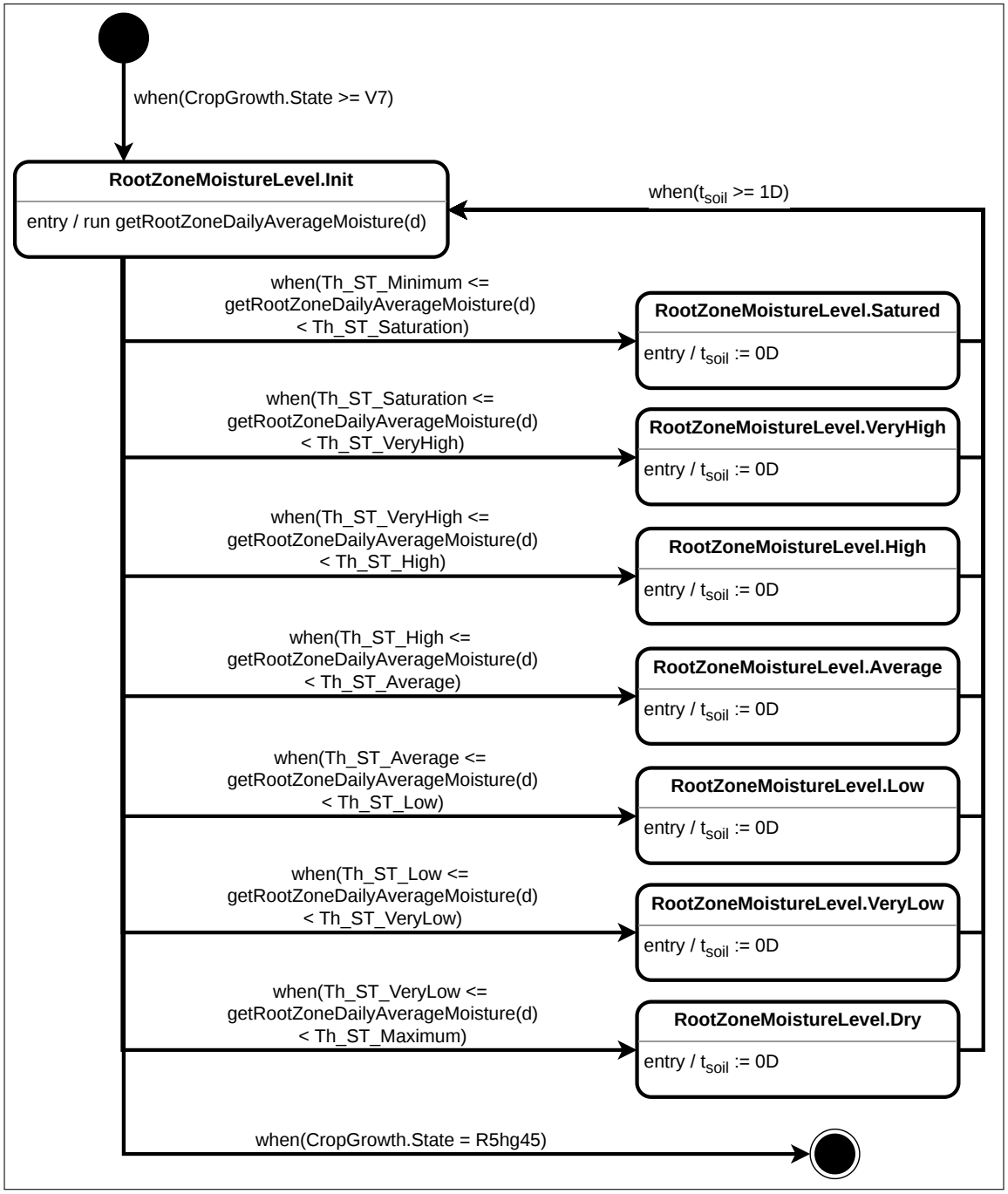



**Figure 44**: Automata of the RootZoneMoistureLevel states and their transitions

or superior to Th_ST_Saturation and inferior to Th_ST_VeryHigh, then the RootZoneMoistureLevel property reaches the `VeryHigh` state. After reaching this state, the clock $t_{soil}$ is reset.

- The transition from the state `Init` towards the state `High` indicates that

when the value of the function getRootZoneDailyAverageMoisture(d) is equal or superior to Th_ST_VeryHigh and inferior to Th_ST_High, then the RootZoneMoistureLevel property reaches the `High` state. After reaching this state, the clock $t_{soil}$ is reset.

- The transition from the state `Init` towards the state `Average` indicates that when the value of the function getRootZoneDailyAverageMoisture(d) is equal or superior to Th_ST_High and inferior to Th_ST_Average, then the RootZoneMoistureLevel property reaches the `Average` state. After reaching this state, the clock $t_{soil}$ is reset.
- The transition from the state `Init` towards the state `Low` indicates when the value of the function getRootZoneDailyAverageMoisture(d) is equal or superior to Th_ST_Average and inferior to Th_ST_Low, then the RootZoneMoistureLevel property reaches the `Low` state. After reaching this state, the clock $t_{soil}$ is reset.
- The transition from the state `Init` towards the state `VeryLow` indicates that when the value of the function getRootZoneDailyAverageMoisture(d) is equal or superior to Th_ST_Low and inferior to Th_ST_VeryLow, then the RootZoneMoistureLevel property reaches the `VeryLow` state. After reaching this state, the clock $t_{soil}$ is reset.
- The transition from the state `Init` towards the state `Dry` indicates that when the value of the function getRootZoneDailyAverageMoisture(d) is equal or superior to Th_ST_VeryLow and inferior to Th_ST_Maximum, then the RootZoneMoistureLevel property reaches the `Dry` state. After reaching this state, the clock $t_{soil}$ is reset.
- The transition from one state among `Saturated`, `VeryHigh`, `High`, `Average`, `Low`, `VeryLow` and `Dry` towards the state `Init` indicates that after one day (($t_{soil}$ $>=$ 1)?), a state change is possible; in other words, a state is reached for a whole day.
- The transition from the state `Init` towards the final state indicates that when the CropGrowth property is at the state `R5hg45`, moisture measurements are terminated.

Figure 45 and Figure 46 present a sample of the RootZoneMoistureLevel deduction on 14/08/2013 to be modeled using CASO and IRRIG.

- An instance of the class **irrig:RootZoneMoistureLevelDeduction** represents a deduction process for the RootZoneMoistureLevel property, as shown in Figure 45. By definition, this individual is linked to the individual irrig:observedProperty_rootZone_moistureLevel, an instance of the class

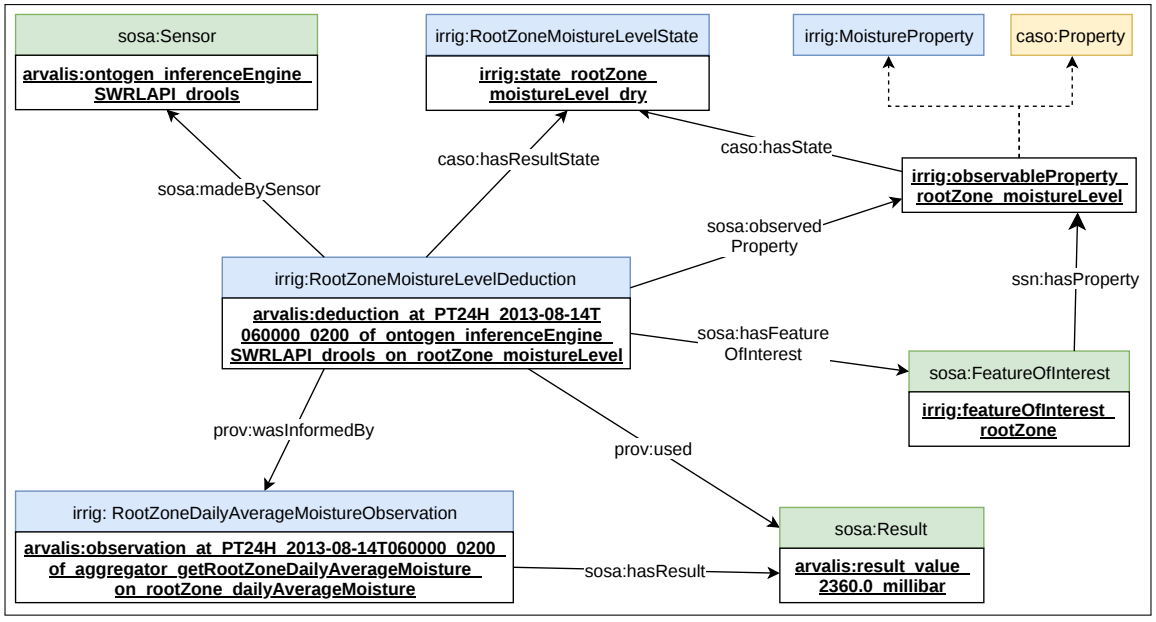



**Figure 45**: RootZoneMoistureLevel deduction on 14/08/2013 (part 1)

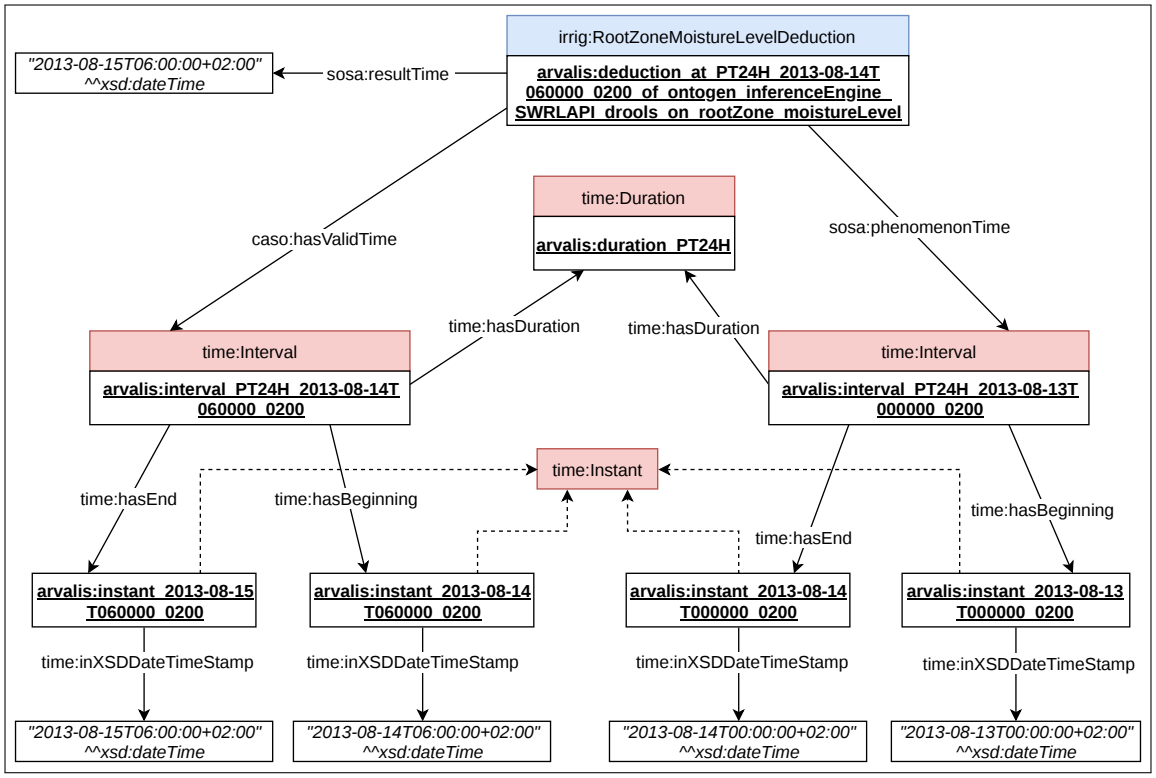



**Figure 46**: RootZoneMoistureLevel deduction on 14/08/2013 (part 2)

**irrig:MoistureProperty**, and to the individual irrig:featureOfInterest_rootZone.

- The deduction individual is linked to an actor that produces the result. The actor is an SWRLAPI Drools Engine that is represented by the individual arvalis:ontogen_inferenceEngine_SWRLAPI_drools, an instance of the class **sosa:Sensor**. The deduction individual is linked to the actor via the `sosa:madeBySensor` object property, as shown in Figure 45.
- A RootZoneMoistureLevel deduction is based on an observation individual that represents a computation of the function `getRootZoneDailyAverageMoisture(d)`. Thus, the deduction instance in Figure 45 is based on the observation individual presented in Figure 42. The deduction individual is linked to the instance of the class **irrig:RootZoneDailyAverageMoistureObservation** via the `prov:wasInformedBy` object property.
- The result of a RootZoneMoistureLevel deduction is an instance of the class **irrig:RootZoneMoistureLevelState**. The deduction individual is linked to its state result by the `caso:hasResultState` object property, as shown in Figure 45. The possible states are shown in Figure 47.

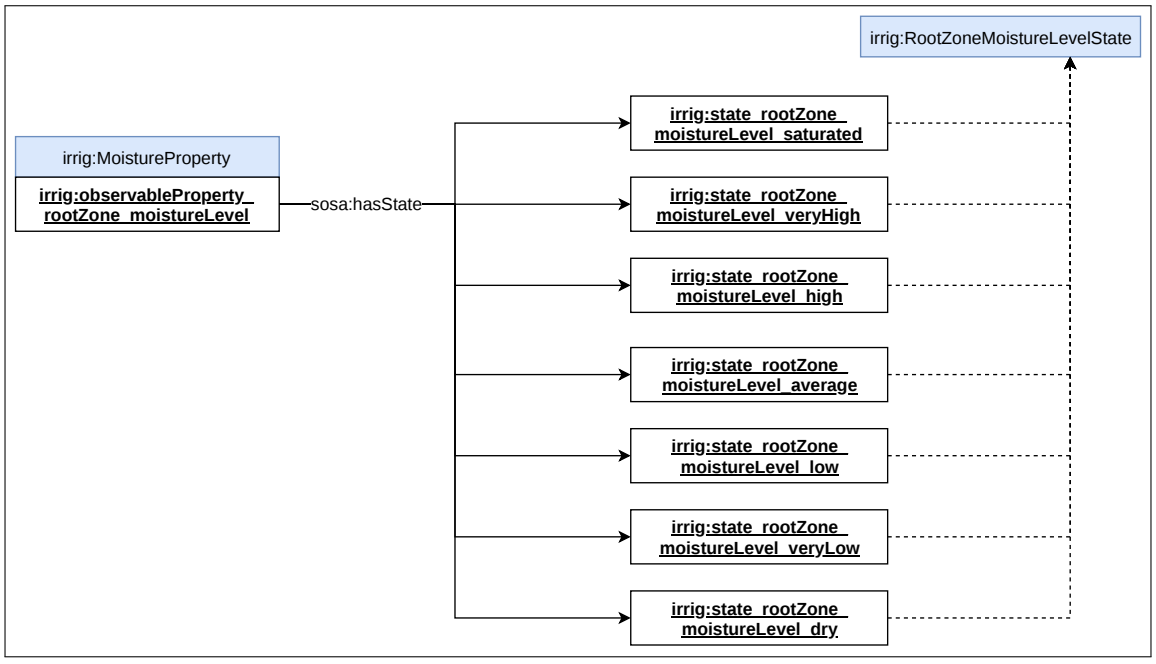



**Figure 47**: States of the property RootZoneMoistureLevel

- To store the time of the deduction, the sosa:resultTime datatype property is used to link the deduction individual to the xsd:dateTime value corresponding to the result time at `[d+1 06:00:00]`. In Figure 46, the result time is at 06:00:00 on 15/08/2013.

- To represent the fact that a deduction concerns the data collected on the day d, the object property `sosa:phenomenonTime` is used. In Figure 46, the day d is on 14/08/2013. It is represented by an instance of the class **time:Interval**, which is described by two instances of the class **time:Instant**. The first one represents the beginning of the interval: `[14/08/2013 00:00:00]`. The second one represents the end of the interval: `[15/08/2013 00:00:00]`.
- The deduction result for day d is valid for 24 h: `[d+1 06:00:00, d+2 06:00:00[`, that is, no new deduction will be performed during this valid time. In Figure 46, the valid time is represented by an instance of the class **time:Interval** for a period of `[15/08/2013 06:00:00, 16/08/2013 06:00:00[`, which is associated with two instances of the class **time:Instant** as presented above. The `caso:hasValidTime` object property links the deduction individual to the valid time.

Figure 48 supports to clarify the synchronization of all the computations of the soil moisture workflow.

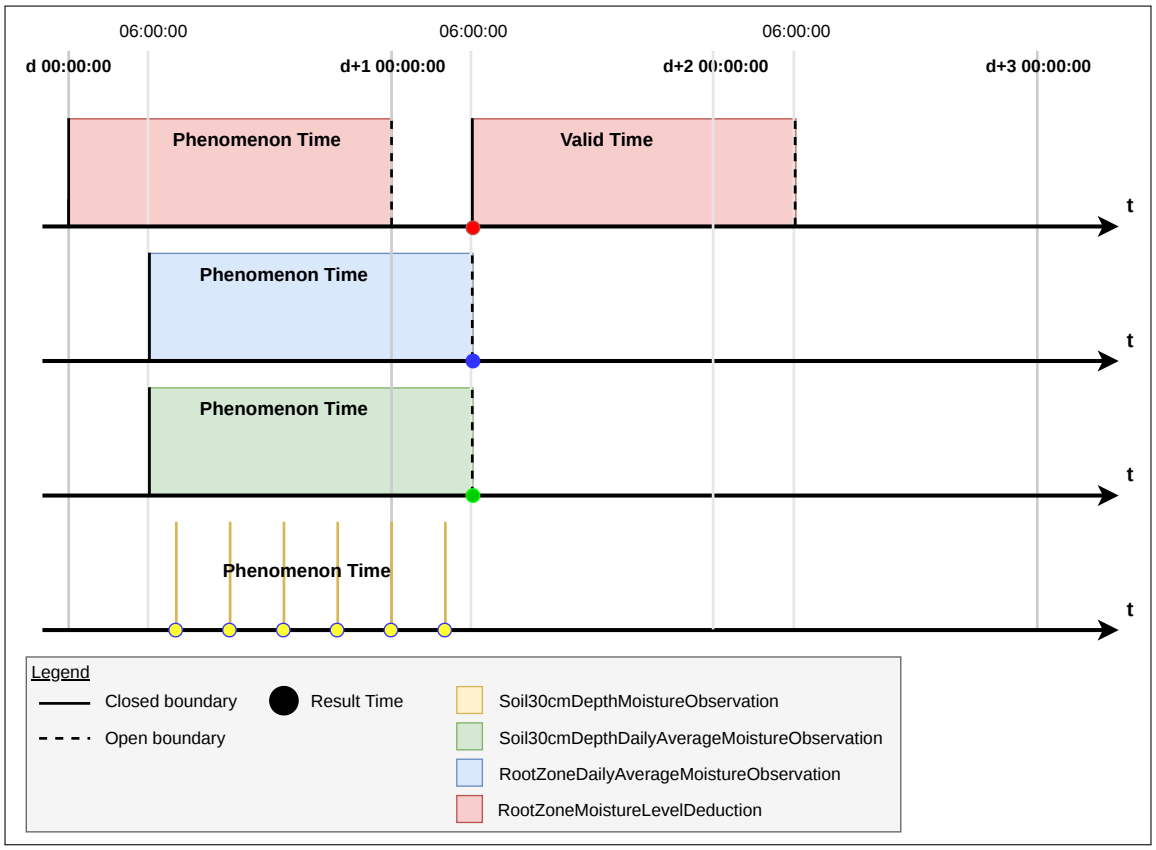



**Figure 48**: Time scales of the soil moisture workflow

#### 3.4.4.5 RootZoneMoistureLevelDeduction Reasoning

Figure 44 gives a global view of the mechanism to deduct a RootZoneMoistureLevel state. In reasoning, one rule can represent this mechanism. This rule is coded in SWRL. Table 13 contains necessary information of the rule. The code in the box **Rule in SWRL** presents the logic of the reasoning process. Detail of the code is as follows.

**Table 13**: Rule for the property RootZoneMoistureLevel to reach a state

| **Code:** ADv122019-RM | **Full name:** ArvalisData-IrrigVersion122019-RootZoneMoistureLevel |
|---|---|
| **Description:** The goal of this rule is to determine the state of RootZoneMoistureLevel. The rule implements the transition from the Init state to one of the state VeryLow, Low, Average, High, VeryHigh and Saturated. The input of this rule is the root zone daily moisture (?result_observation_rootzone_moisture). The output of this rule is the Root Zone Moisture Level deduction (?deduction_rootzone_moisturelevel). The mechanism of this rule is to check if the value of the root zone moisture is in the value domain of two thresholds; it then concludes the state correspondingly. | |
| **Rule in SWRL:**<br>irrig:RootZoneMoistureLevelDeduction(?deduction_rootzone_moisturelevel) ^<br><br>prov:used(?deduction_rootzone_moisturelevel, ?result_observation_rootzone_moisture) ^<br>om:hasNumericalValue(?result_observation_rootzone_moisture, ?value_result_obs) ^<br><br>caso:hasState(irrig:observableProperty_rootZone_moistureLevel,<br>?state_rootzone_moisturelevel) ^<br>caso:hasOpenUpperBoundary(?state_rootzone_moisturelevel, ?boundary_upper) ^<br>caso:boundaryValue(?boundary_upper, ?value_boundary_upper) ^<br>caso:hasClosedLowerBoundary(?state_rootzone_moisturelevel, ?boundary_lower) ^<br>caso:boundaryValue(?boundary_lower, ?value_boundary_lower) ^<br><br>swrlb:lessThan(?value_result_observation, ?value_boundary_upper) ^<br>swrlb:greaterThanOrEqual(?value_result_observation, ?value_boundary_lower)<br><br>-> caso:hasResultState(?deduction_rootzone_moisturelevel, ?state_rootzone_moisturelevel) | |

- The first paragraph is one premise. It means to find the deduction of a root zone moisture level of a day.
- The second paragraph is one premise. It means to get the data value of the root zone moisture of a day.

- The third paragraph is one premise. It means getting the data value of the upper boundary and the lower boundary of every state of root zone moisture level.
- The fourth paragraph is one premise. It means comparing this data value of the root zone moisture observation of a day to check if it is inferior to the value of the upper boundary and superior or equal to the lower boundary of the state of a root zone moisture level.
- The final paragraph is one deduction. It means the deduction of a root zone moisture level has the state satisfy the condition in the fourth paragraph.

### 3.4.5 Crop Water Need Workflow Data Modeling and Reasoning

In the crop water need workflow, data related to crop water need is processed through several processes. The input of this workflow is the states of the three properties CropGrowth, RainIntensity, RootZoneMoistureLevel, and the value of the property DelayDuration. These value are presented in the previous section. The output of this workflow is the state of the property CropWaterNeed and the value of the property SleepingDuration. In detail, the crop water need workflow composes of six processes:

- **CropGrowthDeduction**: see Section 3.4.2.
- **RainIntensityDeductionl**: see Section 3.4.3.
- **DelayDuration**: see Section 3.4.3.
- **RootZoneMoistureLevelDeduction**: see Section 3.4.4.
- **CropWaterNeedDeduction**: a deduction that provides the state of the property CropWaterNeed.
- **SleepingDurationObservation**: a deduction that provides the duration of the feature of interest sleeping. This sleeping duration is calculated based on the delay duration of the day d and the state of CropWaterNeed of the day d-1.

#### 3.4.5.1 CropWaterNeedDeduction Modeling

Crops need water is for two basic biological processes: transpiration and evaporation *(Brouwer et al., 1989)*. The water needs of different crops are different. The water needs of a crop can be expressed in the quantity of water in a unit of time, such as mm/day. On the one hand, the method IRRINOV® answers the question about when to water the plot. On the other hand, it ignores the question about how much water to supply to the crop. In other words, the quantity of each irrigation water

is fixed. Therefore, it requires only to express the crop water needs in Yes/No value each day instead of the quantity of water in a unit of time. This model uses the property CropWaterNeed to represent the crop water need property.

The states of the property CropWaterNeed are deduct based on the states of the three properties RainIntensity, RootZoneMoistureLevel, and CropGrowth. However, this system also concerns the situation that the state of property CropWaterNeed is undefined. In detail, the method IRRINOV® proposes to decide the next irrigation based on an approach of using a time counter to be compared with the parameter called the watering cycle duration. The watering cycle duration is the number of days between two consecutive waterings on the same plot. This interval is modeling as a timed threshold: INTERVAL_WATERING_CYCLE. It is fixed to 7 days as the minimal number of days between two waterings in the Arvalis experimentation. The time counter in this approach starts to count from the last watering. When the counting number is inferior to the interval, then the system makes the watering decision of that day regardless of the state of the property CropWaterNeed. In other words, the state of the property CropWaterNeed, in this case, is undefined. When the counting number is equal or superior to the interval, then the system needs to check the state of the property CropWaterNeed. This approach enables the system to make a watering decision on a day, even the state of the property CropWaterNeed is undefined. To executing the aggregation process to determine the counting number of the time counter, the function `getSleeping(d)` is run. Note that the number of delay days presented in Section 3.4.3.3 has an impact on this function. This function is presented in detail in Section 3.4.5.3.

An inference engine runs this deduction process based on rules. The inference engine is SWRLAPI Drools Engine in the program Ontogen.

The crop water need states are coded as states of the property CropWaterNeed. The list of CropWaterNeed states is as follows:

- **CropWaterNeed.Init**: This state represents the fact that the system waits for a new evaluation of the CropGrowth and RootZoneMoistureLevel states. This state is not an output of the automata. It is a transitional state before determining the new state of the property CropWaterNeed.
- **CropWaterNeed.NotApplicable**: The state of CropWaterNeed is undefined because the system has to wait until the crop growth raises a particular stage.
- **CropWaterNeed.No**: The crop does not need water according to IRRINOV® method.
- **CropWaterNeed.Yes**: The crop needs water according to IRRINOV® method. A watering is launched during the day after this state is reached.

- **CropWaterNeed.Unavailable**: The state of CropWaterNeed is undefined because a new irrigation decision cannot be made due to the minimal time interval between two irrigations. The INTERVAL_WATERING_CYCLE constant represents the minimal number of days between two waterings. Thus, this state specifies that the system waits until at least INTERVAL_WATERING_CYCLE days.

Figure 49 describes that state diagram that presents the inference conception to deduce the state of the property CropWaterNeed based on the states of RainIntensity, CropGrowth, RootZoneMoistureLevel, and the value returned from the function `getSleeping(d)`. In this diagram, there are seven states and seven external transitions. The seven states include two default states of the UML state diagram (initial and final states) and five states defined for the property CropWaterNeed. Among the seven external transitions, only two of them are presented directly in the diagram. The first one is the external transitions from the states `Yes`, `No`, `NotApplicable` and `Unavailable` towards the state Init. This condition means that the state of CropWaterNeed is valid for 24 hours and the system needs to wait for 24 hours before it is re-evaluated. The variable t represents a clock dedicated to the property CropWaterNeed that has a time step of one day. The second one is the external transition from the state `Init` towards the final state. This transition means that when the state of CropGrowth is equal or superior to `R5HG45`, then the system terminated. The other external transitions are complicated, thus they are presented separately in Table 14 under the codes of C0, C1, C2, C3, and C4. Moreover, each state of CropWaterNeed in the diagram contains also several internal transitions.

- The transition from the initial state to the state `Init` occurs on the condition C0. This condition indicates that as soon as the system boots, the CropWaterNeed property reaches the state `Init`. This condition means that the system always starts at the state `Init`. After that, three internal transitions occur:
  1. At the entry, the value of the function `getSleeping(d-1)` is read.
  2. At the entry, the last state of the property CropGrowth is read.
  3. At the entry, the last state of the property RootZoneMoistureLevel is read when the value of the function `getSleeping(d-1)` differs from 0.

  These actions mean that at the state `Init`, the system requires the above information to make the next decisions.
- The transition from the state `Init` to the state `NotApplicable` on the condition C1. This condition indicates that when the value of the property SleepingDuration equals to 0, and the state of the property CropGrowth is inferior to V7, then the property CropWaterNeed reaches the state

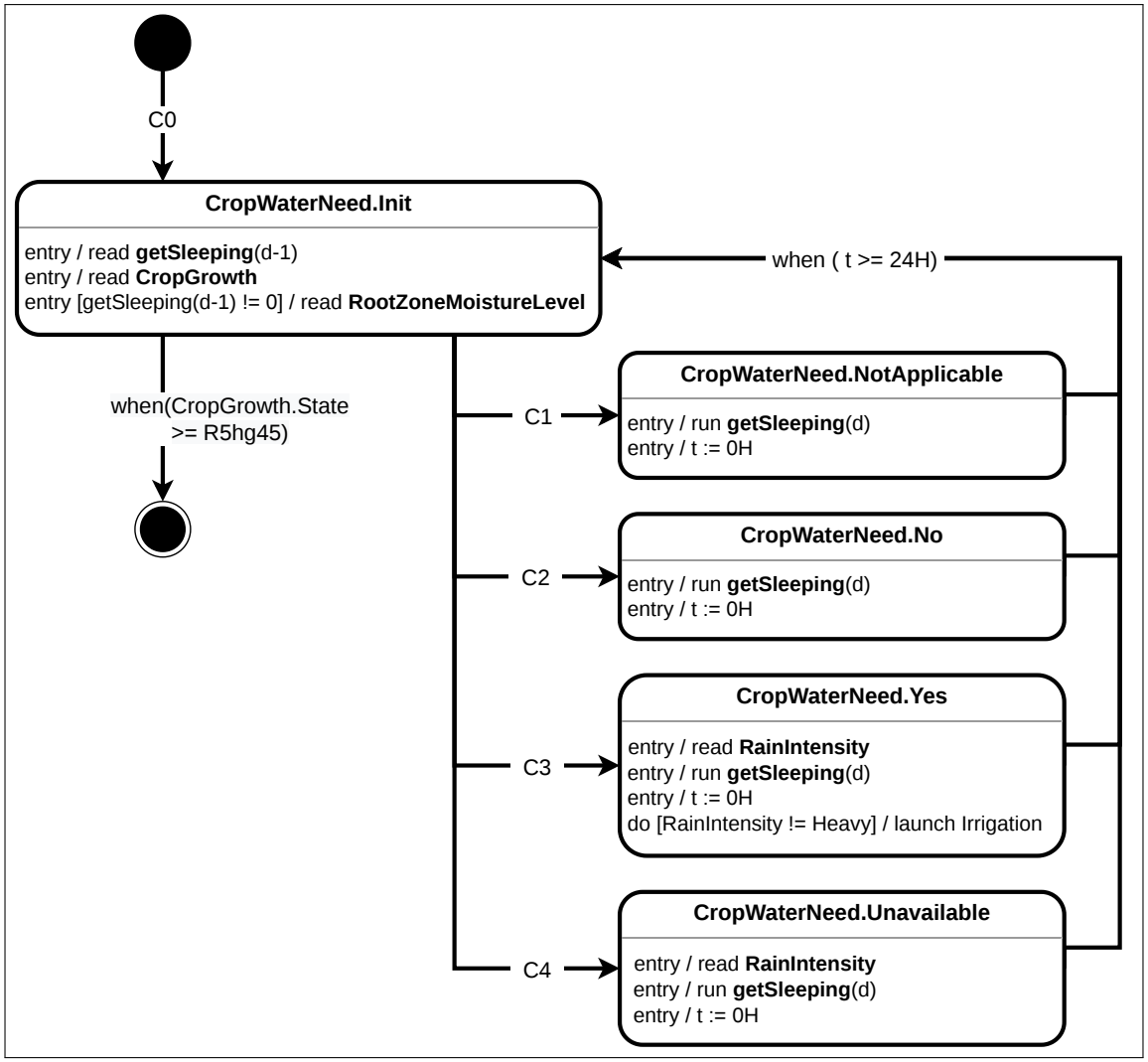



**Figure 49**: Automata of CropWaterNeed states and their transitions

`NotApplicable`. This condition means that the CropWaterNeed is not applicable because the crop growth is immature compared to the state `V7`. After reaching the state `NotApplicable`, two internal transitions occur:

1. At the entry, the function `getSleeping(d)` is run.
2. At the entry, the clock t is reset.

These actions mean that at the state `NotApplicable`, the property CropWaterNeed reaches the state `NotApplicable`, the system calculates the new value of the property SleepingDuration for the day d. In that case, it is fixed to 0 days. The clock t is reset because CropWaterNeed stays at this state for 24 hours.

**Table 14**: Conditions of CropWaterNeed state transitions

| Code | Conditions |
|---|---|
| C0 | SYSTEM_BOOT |
| C1 | when(SleepingDuration = 0 AND CropGrowth.State < V7) |
| C2 | when(SleepingDuration = 0 AND<br>((CropGrowth.State = V7 AND RootZoneMoistureLevel.State > High) OR<br>(CropGrowth.State = V7d20 AND RootZoneMoistureLevel.State > Average) OR<br>(CropGrowth.State = R1 AND RootZoneMoistureLevel.State > Low) OR<br>(CropGrowth.State = R1d15 AND RootZoneMoistureLevel.State > VeryLow) OR<br>(CropGrowth.State = R5 AND RootZoneMoistureLevel.State > Dry))) |
| C3 | when(SleepingDuration = 0 AND<br>(CropGrowth.State = V7 AND RootZoneMoistureLevel.State <= High) OR<br>(CropGrowth.State = V7d20 AND RootZoneMoistureLevel.State <= Average) OR<br>(CropGrowth.State = R1 AND RootZoneMoistureLevel.state <= Low) OR<br>(CropGrowth.State = R1d15 AND RootZoneMoistureLevel.state <= VeryLow) OR<br>(CropGrowth.State = R5 AND RootZoneMoistureLevel.State = Dry))) |
| C4 | when(SleepingDuration > 0) |
| Cfin | when(CropGrowth.State >= R5hg45 |

- The transition from the state `Init` to the state `No` on the condition C2. This condition indicates that when the value of the property SleepingDuration equals to 0, and one among the following conditions is satisfied:
  - The state of CropGrowth is `V7` and the state of RootZoneMoistureLevel is superior to `High`.
  - The state of CropGrowth is `V7d25` and the state of RootZoneMoistureLevel is superior to `Average`.
  - The state of CropGrowth is `R1` and the state of RootZoneMoistureLevel is superior to `Low`.
  - The state of CropGrowth is `R1d15` and the state of RootZoneMoistureLevel is superior to `VeryLow`.
  - The state of CropGrowth is `R5` and the state of RootZoneMoistureLevel is superior to `Dry`,

  then the property CropWaterNeed reaches the state `No`. It means that the state CropWaterNeed reaches the state `No` when the crop growth is mature enough

(superior or equal to V7), and the quantity of water in the root zone is sufficient for the crops. After reaching the state `No`, two internal transitions occur:

1. At the entry, the function `getSleeping(d)` is run.
2. At the entry, the clock t is reset.

These actions mean that at the state `No`, the property CropWaterNeed reaches the state `No`, the system calculates the new value of the property SleepingDuration for the day d. In that case, it is fixed to 0 days. The clock t is reset because CropWaterNeed stays at this state for 24 hours.

- The transition from the state `Init` to the state `Yes` on the condition C3. This condition indicates that when the value of the property SleepingDuration equals to 0, and one among the following conditions is satisfied:
  - The state of CropGrowth is `V7` and the state of RootZoneMoistureLevel is inferior or equal to `High`.
  - The state of CropGrowth is `V7d25` and the state of RootZoneMoistureLevel is inferior or equal to `Average`.
  - The state of CropGrowth is `R1` and the state of RootZoneMoistureLevel is inferior or equal to `Low`.
  - The state of CropGrowth is `R1d15` and the state of RootZoneMoistureLevel is inferior or equal to `VeryLow`.
  - The state of CropGrowth is `R5` and the state of RootZoneMoistureLevel is inferior or equal to `Dry`,

  then the property CropWaterNeed reaches the state `Yes`. It means that the state CropWaterNeed reaches the state `Yes` when the crop growth is mature enough (superior or equal to V7) and the quantity of water in the soil is insufficient for the crops. After reaching the state `Yes`, four internal transitions occur:

  1. At the entry, the state of the property RainIntensity is read.
  2. At the entry, the function `getSleeping(d)` is run.
  3. At the entry, the clock t is reset.
  4. Launch irrigation only when the state of the property RainIntensity is different to `Heavy`.

  These actions mean that at the state `Yes`, the irrigation should be launched if there is no heavy rain during the day d. Then, the system calculates the new value of the property SleepingDuration for the day d. In that case, It is fixed to 6 days. The sleeping period starts. The clock is reset because CropWaterNeed stays at this state for 24 hours.

- The transition from the state `Init` to the state `Unavailable` on the condition C4. This condition indicates that when the value of the property SleepingDuration is superior to 0, the property CropWaterNeed reaches the state `Unavailable`. It means that the state CropWaterNeed reaches the state `Unavailable` when the sleeping period has started and is not finished. After reaching the state `Unavailable`, three internal transitions occur:
  1. At the entry, the state of the property RainIntensity is read.
  2. At the entry, the function `getSleeping(d)` is run.
  3. At the entry, the clock t is reset.

  These actions mean that at the state `Unavailable`, the system needs to know if it was raining during the day d to calculate the new value of the property SleepingDuration for this day. The sleeping period may vary from 1 to 15 days, depending on situations. The clock is reset because CropWaterNeed stays at this state for 24 hours.

Figure 50 and Figure 51 present a sample of the CropWaterNeed deduction on 14/08/2013 to be modeled using CASO and IRRIG.

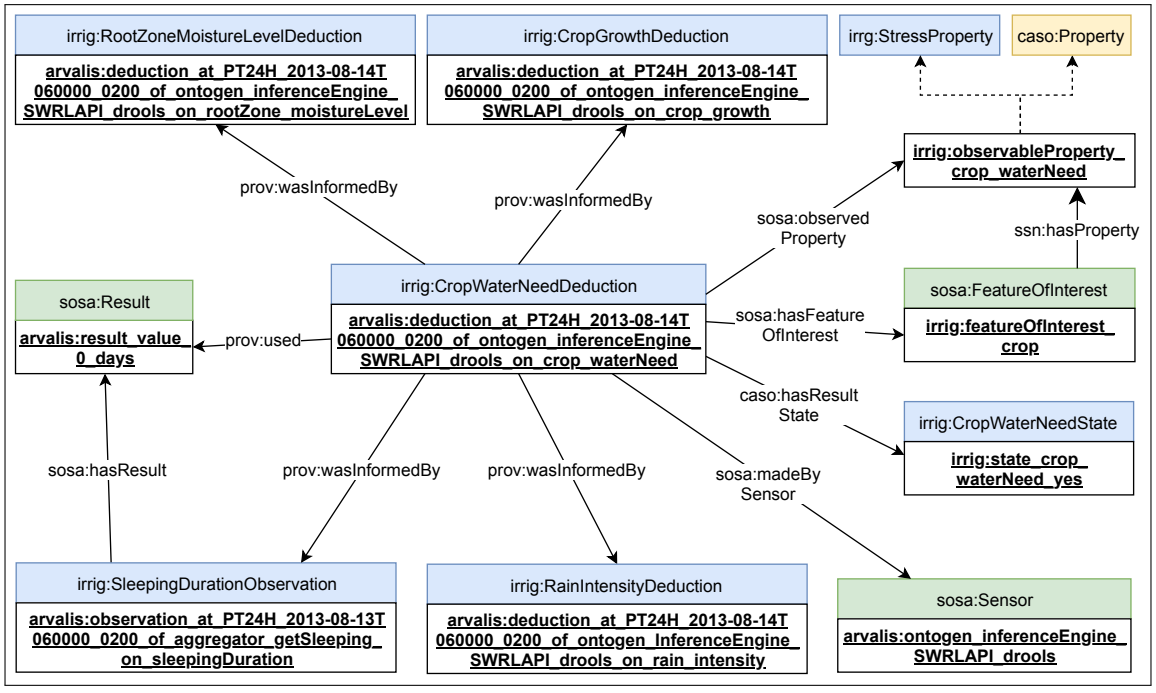



**Figure 50**: CropWaterNeed deduction on 14/08/2013 (part 1)

- An instance of the class **irrig:CropWaterNeedDeduction** represents a deduction process for the property CropWaterNeed, as shown in Figure 50. By definition, this individual is linked to the individual

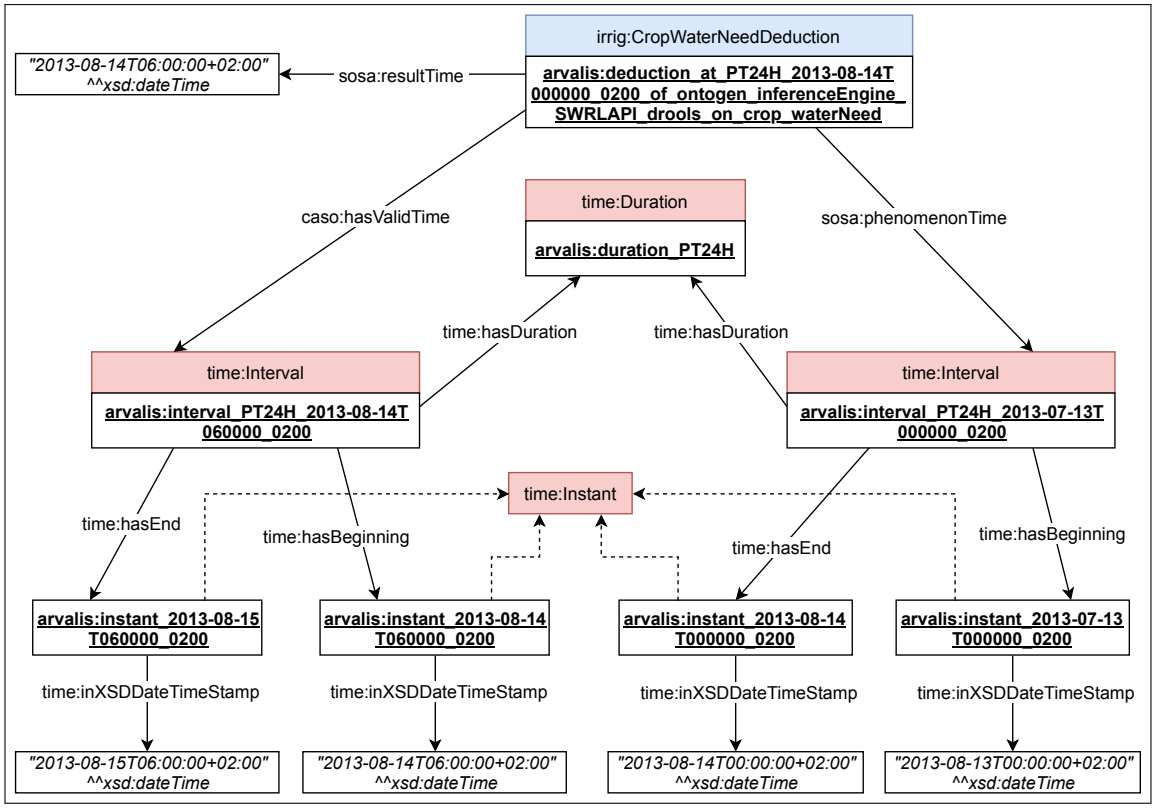



**Figure 51**: CropWaterNeed deduction on 14/08/2013 (part 2)

irrig:observedProperty_crop_waterNeed, an instance of the class **irrig:StressProperty**, and to the individual irrig:featureOfInterest_crop, an instance of the class **sosa:FeatureOfInterest**.

- The deduction individual is linked to an actor that produces the result. The actor is the SWRLAPI Drools Engine which is represented by the individual arvalis:ontogen_inferenceEngine_SWRLAPI_drools, an instance of the class **sosa:Sensor**. The deduction individual is linked to the actor via the `sosa:madeBySensor` object property, as shown in Figure 50.
- A CropWaterNeed deduction is based on the deductions of RootZoneMoistureLevel, RainIntensity, CropGrowth, and the observation of SleepingDuration. Thus, the deduction individual is linked to the instances of the classes **irrig:RootZoneMoistureLevelDeduction**, **irrig:CropGrowthDeduction**, **irrig:RainIntensityDeduction** and **irrig:SleepingDurationObservation** via the `prov:wasInformedBy` object property. Also, the deduction individual is linked to the result of **irrig:SleepingDurationObservation** via the `prov:used` object property.
- The result of a CropWaterNeed deduction is an instance of the

**irrig:CropWaterNeedState** class. The deduction individual is linked to its state result by the `caso:hasResultState` object property, as shown in Figure 50. The possible states are shown in Figure 52.

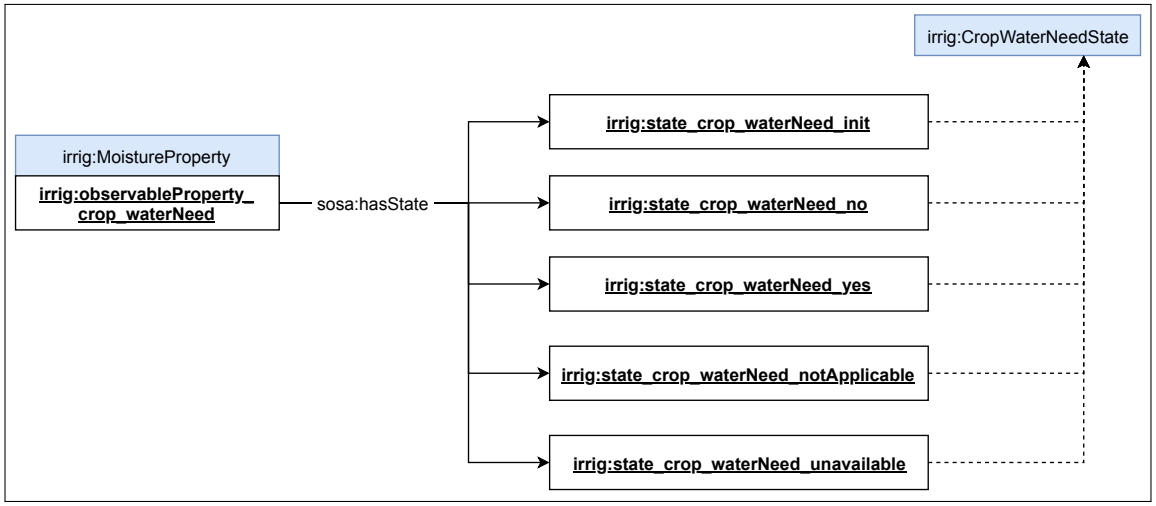



**Figure 52**: States of the property CropWaterNeed

- To store the time of the deduction, the sosa:resultTime datatype property is used to link the deduction individual to the xsd:dateTime value corresponding to the result time at `[d+1 06:00:00]`. In Figure 51, the result time is at 06:00:00 on 15/08/2013.
- To represent the fact that a deduction concerns the data collected on the day d, the object property `sosa:phenomenonTime` is used. In Figure 51, the day d is on 14/08/2013. It is represented by an instance of the class **time:Interval**, which is described by two instances of the class **time:Instant**. The first one represents the beginning of the interval: `[14/08/2013 00:00:00]`. The second one represents the end of the interval: `[15/08/2013 00:00:00]`. The `sosa:phenomenonTime` object property links the deduction individual to the interval individual.
- The deduction result for day d is valid for 24 h: `[d+1 06:00:00, d+2 06:00:00[`, that is, no new deduction will be performed during this valid time. It also means that the farmers can water the farmland in this valid period. In Figure 51, the valid time is represented by an instance of the class **time:Interval** for a period of `[15/08/2013 06:00:00, 16/08/2013 06:00:00[`, which is associated with two instances of the class **time:Instant** as presented above. The `caso:hasValidTime` object property links the deduction individual to the valid time.

Figure 53 supports to clarify the synchronization of all the computations of the crop water need workflow.

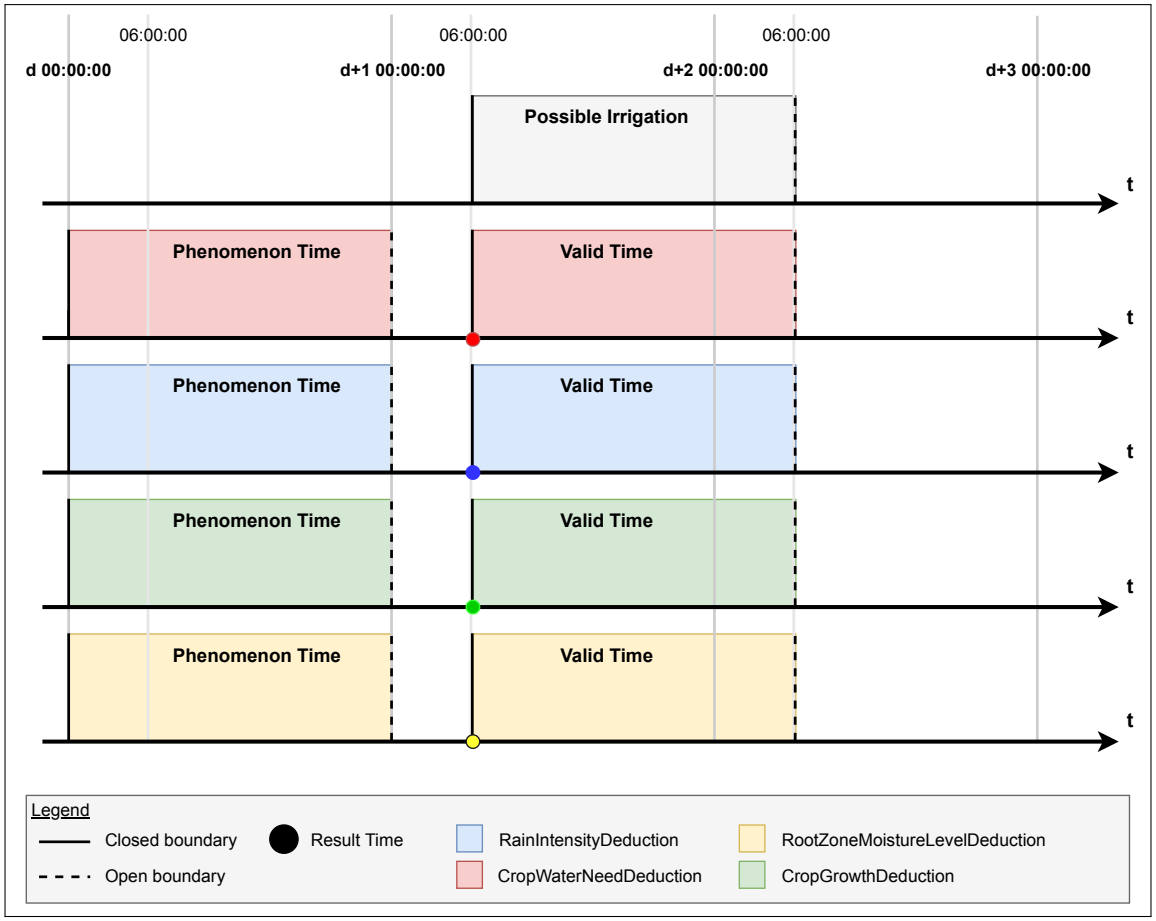



**Figure 53**: Time scales of the crop water need workflow

#### 3.4.5.2 CropWaterNeedDeduction Reasoning

Figure 49 gives a global view of the mechanism to deduct a CropWaterNeed state. In reasoning, this mechanism is transformed into 12 rules corresponding to the four conditions C1, C2, C3, and C4, as presented in Table 14. Of which, one rule represents the condition C1 that the CropWaterNeed reaches the state `NotApplicable`. One rule represents the condition C4 that the CropWaterNeed reaches the state `No`. Five rule represents the condition C2 that the CropWaterNeed reaches the state `Yes`. Five rules represent the condition C3 that the CropWaterNeed reaches the state `Unavailable`. All of the rules are coded in SWRL. This subsection presents four rules corresponding to the four conditions. Note that the five rules for C2 have the same concept but different parameters. Thus, it is only necessary to present only one rule for C2. Same to the case of C3.

Table 15 contains necessary information of the rule for the condition C1. The code in the box **Rule in SWRL** presents the logic of the reasoning process. Detail

of the code is as follows.

**Table 15**: Rule for the property CropWaterNeed to reach the NonApplicable state

| **Code:** | **Full name:** |
|---|---|
| ADv122019-CWN-C1 | ArvalisData-IrrigVersion122019-CropWaterNeed-Condition-1 |

**Description:**
The goal of this rule is to determine the state of CropWaterNeed. The rule implements the transition from the Init state to the NonApplicable. The input of this rule is: (1) the value of the SleepingDuration property for the d-1 day (?sleeping_duration_yesterday); (2) the state of the CropGrowth property for the d day (?state_crop_growth). The output of this rule is: the CropWaterNeed property is at the state NotApplicable for the d day. The mechanism of this rule is: to check whether the state of CropGrowth is inferior to V7 using the caso:lesserThan observable property. To check the value of SleepingDuration is equal to 0 by verifying if it exists a link between the result of the SleepingDuration observation and the arvalis:result_value_0_days individual.

**Rule in SWRL:**

```
irrig:CropWaterNeedDeduction(?deduction_crop_water_need) ^

irrig:SleepingDurationObservation(?observation_sleeping_duration_yesterday) ^
prov:wasInformedBy(?deduction_crop_water_need,
                                        ?observation_sleeping_duration_yesterday) ^
sosa:hasResult(?observation_sleeping_duration_yesterday, arvalis:result_value_0_days) ^

irrig:CropGrowthDeduction(?deduction_crop_growth) ^
prov:wasInformedBy(?deduction_crop_water_need, ?deduction_crop_growth) ^
caso:hasResultState(?deduction_crop_growth, ?state_crop_growth) ^
caso:lesserThan(?state_crop_growth, irrig:state_crop_growth_v7)

-> caso:hasResultState(?deduction_crop_water_need,
                                        irrig:state_crop_waterNeed_notApplicable)
```

- The first paragraph is one premise. It means to find every crop water need deduction.
- The second paragraph is one premise. It means to select the crop water need deduction that has a sleeping duration value equaling to 0 days.
- The third paragraph is one premise. It means to select the crop water need deduction that was informed by a crop growth deduction at the state inferior to `V7`.
- The final paragraph is one deduction. It means the crop water need deduction is at the state `NotApplicable`.

Table 16 contains necessary information of the rule for the first part of the condition C2. The code in the box **Rule in SWRL** presents the logic of the reasoning

process. Detail of the code is as follows.

**Table 16**: Rule for the property CropWaterNeed to reach the No state (part 1)

| **Code:** | **Full name:** |
|---|---|
| ADv122019-CWN-C2-P1 | ArvalisData-IrrigVersion122019-CropWaterNeed-Condition-2-Part-1 |
| **Description:** The goal of this rule is to determine the state of CropWaterNeed. The rule implements the transition from the Init state to the NonApplicable. The input of this rule is: (1) the value of the SleepingDuration property of d-1 (?sleeping_duration_yesterday); (2) the state of the CropGrowth property for the d day (?state_crop_growth); (3) the state of RootZoneMoistureLevel for the d day. The output of this rule is: the CropWaterNeed property is at the state No for the d day. The mechanism of this rule is: to check if the value of SleepingDuration is 0, the state of CropGrowth is V7 and the state of RootZoneMoistureLevel is superior to High. | |
| **Rule in SWRL:**<br>irrig:CropWaterNeedDeduction(?deduction_crop_water_need) ^<br><br>irrig:SleepingDurationObservation(?observation_sleeping_duration_yesterday) ^<br>prov:wasInformedBy(?deduction_crop_water_need,<br>?observation_sleeping_duration_yesterday) ^<br>sosa:hasResult(?observation_sleeping_duration_yesterday, arvalis:result_value_0_days) ^<br><br>irrig:CropGrowthDeduction(?deduction_crop_growth) ^<br>prov:wasInformedBy(?deduction_crop_water_need, ?deduction_crop_growth) ^<br>caso:hasResultState(?deduction_crop_growth, irrig:state_crop_growth_v7) ^<br><br>irrig:RootZoneMoistureLevelDeduction(?deduction_rootzone_moisturelevel) ^<br>prov:wasInformedBy(?deduction_crop_water_need, ?deduction_rootzone_moisturelevel) ^<br>caso:hasResultState(?deduction_rootzone_moisturelevel, ?state_rootzone_moisturelevel) ^<br>caso:greaterThan(?state_rootzone_moisturelevel, irrig:state_rootZone_moistureLevel_high)<br><br>-> caso:hasResultState(?deduction_crop_water_need, irrig:state_crop_waterNeed_no) | |

- The first paragraph is one premise. It means to find every crop water need deduction.
- The second paragraph is one premise. It means to select the crop water need deduction that has a sleeping duration value equaling to 0 days.
- The third paragraph is one premise. It means to select the crop water need deduction that was informed by a crop growth at the state `V7`.
- The fourth paragraph is one premise. It means to select the crop water need deduction that was informed by a root zone moisture level at the state superior to `High`.

- The final paragraph is one deduction. It means the crop water need deduction is at the state `No`.

Table 17 contains necessary information of the rule for the first part of the condition C3. The code in the box **Rule in SWRL** presents the logic of the reasoning process. Detail of the code is as follows.

**Table 17**: Rule for the property CropWaterNeed to reach the Yes state (part 1)

| **Code:** | **Full name:** |
|---|---|
| ADv122019-CWN-C3-P1 | ArvalisData-IrrigVersion122019-CropWaterNeed-Condition-3-Part-1 |

**Description:**
The goal of this rule is to determine the state of CropWaterNeed. The rule implements the transition from the Init state to the Yes. The input of this rule is: (1) the value of the SleepingDuration property of d-1 (?sleeping_duration_yesterday); (2) the state of the CropGrowth property for the d day (?state_crop_growth); (3) the state of RootZoneMoistureLevel for the d day. The output of this rule is: the CropWaterNeed property is at the state Yes for the d day. The mechanism of this rule is: to check if the value of SleepingDuration is 0, the state of CropGrowth is V7 and the state of RootZoneMoistureLevel is inferior or equal to High.

**Rule in SWRL:**

```
irrig:CropWaterNeedDeduction(?deduction_crop_water_need) ^

irrig:SleepingDurationObservation(?observation_sleeping_duration_yesterday) ^
prov:wasInformedBy(?deduction_crop_water_need,
                                        ?observation_sleeping_duration_yesterday) ^
sosa:hasResult(?observation_sleeping_duration_yesterday, arvalis:result_value_0_days) ^

irrig:CropGrowthDeduction(?deduction_crop_growth) ^
prov:wasInformedBy(?deduction_crop_water_need, ?deduction_crop_growth) ^
caso:hasResultState(?deduction_crop_growth, irrig:state_crop_growth_v7) ^

irrig:RootZoneMoistureLevelDeduction(?deduction_rootzone_moisturelevel) ^
prov:wasInformedBy(?deduction_crop_water_need, ?deduction_rootzone_moisturelevel) ^
caso:hasResultState(?deduction_rootzone_moisturelevel, ?state_rootzone_moisturelevel) ^
caso:lesserThanOrEqualTo(?state_rootzone_moisturelevel,
                                        irrig:state_rootZone_moistureLevel_high)

-> caso:hasResultState(?deduction_crop_water_need, irrig:state_crop_waterNeed_yes)
```

- The first paragraph is one premise. It means to find every crop water need deduction.
- The second paragraph is one premise. It means to select the crop water need deduction that has a sleeping duration value equaling to 0 days.

- The third paragraph is one premise. It means to select the crop water need deduction that was informed by a crop growth at the state `V7`.
- The fourth paragraph is one premise. It means to select the crop water need deduction that was informed by a root zone moisture level at the state inferior or equal to `High`.
- The final paragraph is one deduction. It means the crop water need deduction is at the state `Yes`.

Table 18 contains necessary information of the rule for the condition C4. The code in the box **Rule in SWRL** presents the logic of the reasoning process. Detail of the code is as follows.

**Table 18**: Rule for the property CropWaterNeed to reach the Unavailable state

| **Code:** ADv122019-CWN-C4 | **Full name:** ArvalisData-IrrigVersion122019-CropWaterNeed-Condition-4 |
|---|---|
| **Description:** The goal of this rule is to determine the state of CropWaterNeed. The rule implements the transition from the Init state to the Unavailable. The input of this rule is: the value of the SleepingDuration property of yesterday (?sleeping_duration_yesterday). The output of this rule is: the CropWaterNeed property is at the Unavailable state. The mechanism of this rule is: to check if the value of SleepingDuration is superior to 0 using the built-in function swrlb:greaterThan to compare the numerical value of the result of SleepingDuration with 0. | |
| **Rule in SWRL:**<br>irrig:CropWaterNeedDeduction(?deduction_crop_water_need) ^<br><br>irrig:SleepingDurationObservation(?observation_sleeping_duration_yesterday) ^<br>prov:wasInformedBy(?deduction_crop_water_need,<br>?observation_sleeping_duration_yesterday) ^<br>sosa:hasResult(?observation_sleeping_duration_yesterday, ?sleeping_duration_yesterday) ^<br>om:hasNumericalValue(?sleeping_duration_yesterday, ?value_sleeping_duration) ^<br>swrlb:greaterThan(?value_sleeping_duration, 0)<br><br>-> caso:hasResultState(?deduction_crop_water_need, irrig:state_crop_waterNeed_unavailable) | |

- The first paragraph is one premise. It means to find every crop water need deduction.
- The second paragraph is one premise. It means to select the crop water need deduction that has a sleeping duration value superior to 0 days.
- The final paragraph is one deduction. It means the crop water need deduction is at the state `Unavailable`.

#### 3.4.5.3 SleepingDurationObservation Modeling

This watering cycle duration is decomposed into two periods: an irrigation period and a sleeping period (period wait until the next irrigation). In the Arvalis experimentation: the irrigation period equals to 1 day, and the minimal sleeping period is fixed to 6 days. The sleeping period or the number of sleeping days is produced after an aggregation process. To execute this process, the function `getSleeping(d)` is run. Table 19 presents this function. The SleepingDuration observable property expresses the duration of the sleeping period. The sleeping period is initialized when CropWaterNeed property reaches the state `Yes`. Then, the value of SleepingDuration will decrease each day when CropWaterNeed state is at the state `Unavailable`. Otherwise, SleepingDuration is fixed at 0. The value of SleepingDuration is impacted by the rain quantity, due to the fact than rain extends the sleeping period. Thus, the value of SleepingDuration depends on the result of the function `getDelay(d)` only when CropWaterNeed is at the state `Unavailable`. Thus, the duration of the sleeping period is calculated each time the CropWaterNeed property reaches a state.

**Table 19**: Function getSleeping(d)

$$getSleeping(d) = \begin{cases} MIN(getSleeping(d-1) + getDelay(d) - 1, \\ \quad \textbf{MAX\_DELAY} + \textbf{WATERING\_CYCLE\_DURATION}) \\ \qquad \text{if CropWaterNeed} = \textbf{Unavailable} \\ \textbf{WATERING\_CYCLE\_DURATION} - 1 \\ \qquad \text{if CropWaterNeed} = \textbf{Yes} \\ 0 \qquad \text{if CropWaterNeed} = \textbf{NotApplicable} \text{ or } \textbf{No} \end{cases}$$

where

- d: a specific day, e.g., 14/08/2013
- WATERING_CYCLE_DURATION: is a constant that expresses the minimum duration between two waterings. In Arvalis experiment it is fixed to 7 days.
- MAX_DELAY: is a constant that represents the maximum duration of the delay period. We fix this constant to 8 days.
- getSleeping(d): is the duration of the sleeping period associated to d.
- getDelay(d): is the duration of the delay period associated with the day d.
- MIN(x,y): is the smaller value between x and y.

Figure 54 presents a sample of the SleepingDuration observation on 14/08/2013 to be modeled using CASO and IRRIG.

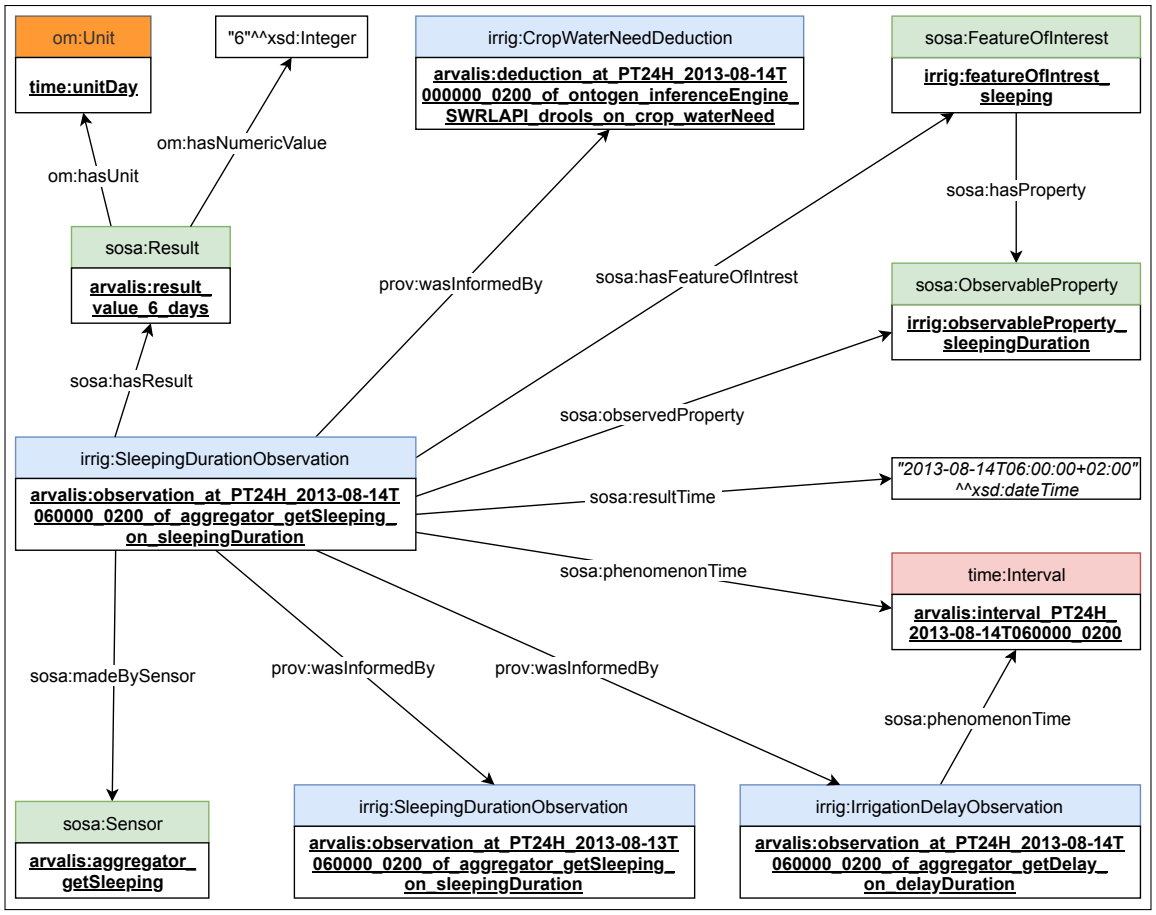



**Figure 54**: SleepingDuration observation on 14/08/2013

- The instance of the class **irrig:SleepingDurationObservation** represents the computation of the function `getSleeping(d)`, as shown in Figure 54. By definition, this individual is linked to the indiviudal irrig:observedProperty_sleepingDuration, an instance of the class **sosa:ObservableProperty**, and to the individual irrig:featureOfInterest_sleeping, an instance of the class **sosa:FeatureOfInterest**.
- The observation individual is linked to the actor that produces the result, that is, an aggregator runs the function `getSleeping(d)`. Since there is no information about the aggregator that produces this computation in the Arvalis dataset, it is possible to represent it by the individual arvalis:aggregator_getSleeping, an instance of the class **sosa:Sensor**. The observation individual is linked to the sensor individual via the `sosa:madeBySensor` object property, as show in Figure 54.

- The aggregator produces the result once per day at a precise time. To present this time, the sosa:resultTime datatype property is used to link the observation individual to the `xsd:dateTime` value. In the Arvalis dataset, there is no information about the hour, minute, and second of the computation. For synchronization purposes, this precise time is fixed to the time of the deduction computation `[d+1 06:00:00]`. In Figure 54, this precise time is at 06:00:00 on 15/08/2013.
- To represent the fact that an observation concerns the data collected on the day d, the object property `sosa:phenomenonTime` is used. In Figure 54, the phenomenon time is represented by an instance of the class **time:Interval** for a period of `[14/08/2013 06:00:00, 15/08/2013 06:00:00[`, which is associated with two instances of the class **time:Instant**. The `sosa:phenomenonTime` object property links the deduction individual to the interval individual.
- The computation of the function `getSleeping(d)` function is based on three actions: (1) the deduction of the property CropWaterNeed, (2) the computation of the getSleeping(d-1) function, and (3) the computation of the function `getDelay(d)`. Then, this observation individual is linked to the three actions via the object property `prov:wasInformedBy`, as shown in Figure 54.
- The result of the computation is represented by an instance of the class **sosa:Result**. This instance is described by a unit and a value, as shown in Figure 54.

### 3.4.6 Services of the Irrigation Decision Support System in TSCF

In theory, the stack of services of the irrigation CAS in TSCF must consist of services from all of the four phases of the context life cycle of CASs. Since the irrigation DSS presented in this subsection is part of the irrigation CAS in TSCF, this DSS provides only services in the modeling phase and analysis phase. Figure 55 presents the services run by the DSS. There are a total of six services as follows.

- **Annotation**: This service equals to the integration process. It models the input data using IRRIG. The input data is Arvalis data stored in an excel document.
- **Storage**: This service equals to both long-term storage and short-term storage processes. After being modeled, it stores the data into owl files or temporarily saves data in the computer's RAM (Random-Access Memory). The former case is long-term storage, and the latter case is short-term storage.

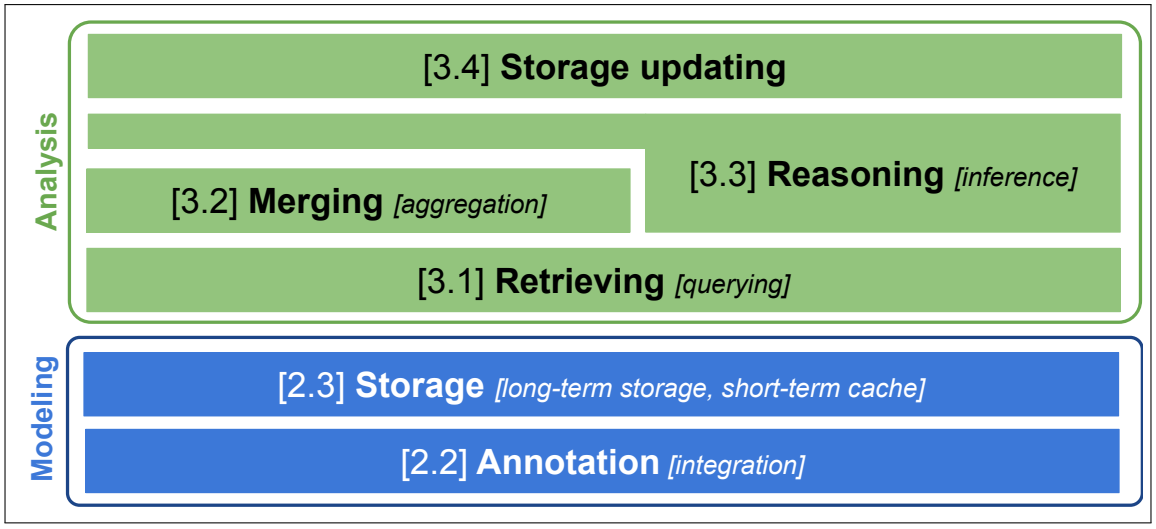



**Figure 55**: Services run by the irrigation DSS in TSCF

- **Retrieving**: This service equals to querying process. It queries data from the long-term storage or short-term storage of the DSS.
- **Merging**: This service equals to aggregation process. It produces aggregate data from the retrieved data through some functions as presented in the previous subsections.
- **Reasoning**: This service equals to inference process. It produces deducted data, which are states of the entities of the four workflows.
- **Storage Updating**: This service distributes the new data into the storage.

## 3.5 System Implementation

Ontogen is a software program developed specialized for the DSS module of the CAS irrigation system in TSCF. It is coded in Python and Java. The first version of Ontogen (`Ontogen-v201912`) corresponds to the first version of the CAS irrigation system. Ontogen takes as input sensor measurements to produce the state of the property CropWaterNeed as output. It is divided into three modules, as follows.

- Virtual sensor: A module can read data from a CSV file or receive messages. The former case is when all recorded data are in a file. The latter case is in real experimentation.
- Aggregator: A module run functions to produce aggregate data from measured data. The version `Ontogen-v201912` consists of four functions which are `getDelay`, `getRainDailyTotalQuantity`, `getDailyAggergateMoisture`, and

`getRootZoneDailyAverageMoisture`.

- SWRLAPI Drools Inference engine: A module runs Drools inference engine. The core of this inference engine is OWLAPI[8] and SWRLAPI[9]. This module uses a set of SWRL rules and all the data provided from the previous module to deduct high-level context data.

The aggregations are encoded in Python as functions. The deductions are executed by the SWRLAPI Drool inference engine integrated into Ontogen. In detail, the reasoning process of the inference engine in Ontogen can be explained as follows. First, Ontogen generates an OWL file that contains all the individuals and the IRRIG ontology necessary for the rule engine; it creates all the individuals that describe sensor measurements. It also creates the individuals, instances of **caso:Deduction**, necessary to define the daily deduction per instance of the class **caso:Property**. The file name is the concatenation of the feature of interest name and the beginning date. Second, it launches the Drools rule engine and applies it to the individuals previously created. The Drools engine updates the **caso:Deduction** instance. It links an instance of the class **caso:State** with an instance of **caso:Deduction** via the object property `caso:hasResultState`. Then, it creates a new OWL file that contains the results of the inference engine. The OWL file is updated from the previous file. The rule for Drools is coded in SWRL. There are a total of 24 rules in SWRL as follows.

- 6 rules for crop growth.
- 1 rule for rain intensity.
- 1 rule for root zone soil moisture level.
- 16 rules for crop water need.

The contents of the rules are presented in Appendix A. Each rule is put in a rule file with a `.rule` extension. The rule files are available on the GitLab repository of INRAE via the following link https://gitlab.irstea.fr/irrig/public/tree/master/Rule.

Figure 56 shows the algorithm implemented by the DSS using the Ontogen program. It is the case that the virtual sensor module read data from the Arvalis dataset, as presented in Section 3. At 06:00:00 on day d, the system triggers the DSS to start the computation. First, the DSS retrieves the observations relevant to days d-1 and d recorded in the Arvalis dataset. Note that in the algorithm of Figure 56, the data retrieved at the instant `[d 06:00:00]` belong to the period `[d-1 00:00:00, d 06:00:00]`. The DSS checks the value of the evaluation of the **CropGrowth** property of day d-1. As the method IRRINOV® is valid only when the state of **CropGrowth** is from `V7` to `R5hg45`, then the DSS has three possible cases: (1) if

[8] https://github.com/owlcs/owlapi

[9] https://github.com/protegeproject/swrlapi

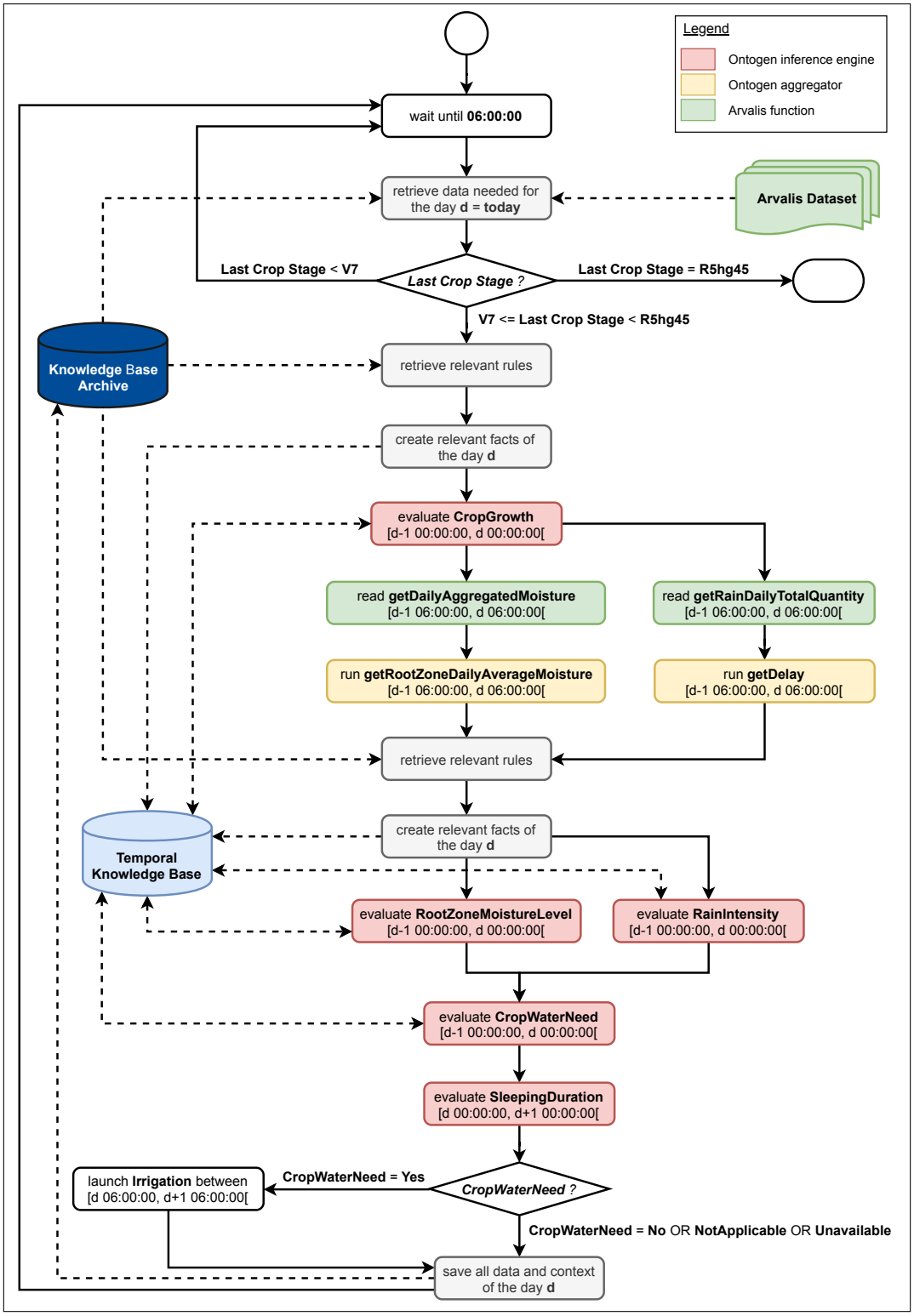



**Figure 56**: Algorithm in Ontogen for the DSS

the state of **CropGrowth** is less than `V7`, then the DSS waits until the next day to reevaluate the **CropGrowth**, (2) if the state of **CropGrowth** equals `R5hg45`, then the DSS stops, and (3) if the state of **CropGrowth** is between `V7` and `R5hg45`, then the DSS continues to retrieve rules to create a fact related to **CropGrowth**. The fact integrates rules as axioms. After determining the fact, the DSS can launch the Drools inference engine to infer the state of **CropGrowth** for day d-1. Next, the DSS runs the functions to obtain the value of the root zone average moisture of day d-1 and to obtain the number of delay days. The function to obtain the value of root zone average moisture takes the value of the average moisture of day d-1 corresponding to each soil tensiometer from the Arvalis dataset. Additionally, the function to obtain the number delay days takes the value of the total rain quantity of day d-1 from the Arvalis dataset. The DSS collects all calculated results and retrieves the other rules related to **RainIntensity**, **RootZoneMoistureLevel** and **CropWaterNeed** properties and updates them. Later, the Drools inference engine infers the state of the three mentioned properties and the value of **SleepingDuration**. Based on the state of **CropWaterNeed**, the DSS offers two solutions: (1) if the state of **CropWaterNeed** is `Yes`, the DSS suggests that users launch irrigation on day d; otherwise, (2) if the state of **CropWaterNeed** is `No`, `NotApplicable` or `Unavailable`, the DSS suggests that users do nothing. After producing the suggestion, the system saves the results of all computations into the knowledge base. Finally, the DSS continues to wait until 06:00:00 of the next day (d+1).

## 3.6 System Testing

The system testing step is to evaluate systems. The DSS works if it passes all the test cases without faults, or at least the faults are acceptable and explainable. System testing is not only for the DSS, but also it indirectly to evaluate the correctness of the two ontologies CASO and IRRIG. This research proposes three levels of system testing as follows.

- Unit test: To check whether a rule can provide expected results as testers' calculations. Testers deliberately choose the input value.
- Sample test: To check whether a rule can provide expected results as the values of an real experiment.
- System test: To check whether the system using all of the rules provides expected results as the reasoning of system developers based on data from a real experiment.

### 3.6.1 Evaluation Using Unit Test

This research performed a total of 120 unit test cases for the four workflows.

- 13 test cases for crop growth workflow.
- 11 test cases for rain intensity workflow.
- 13 test cases for soil moisture workflow.
- 83 test cases for crop water need workflow.

In each test case, the system takes one or several input values depending on the corresponding workflow and produces one output result. It is a deduction process where the SWRLAPI Drools inference engine applies each rule to the knowledge base, including the input value, to infer a deducted result. It is possible to evaluate the rule by comparing the deducted result with the expected result calculated by stakeholders. The information related to the comparisons is available in the different tables corresponding to workflows. Each table contains several columns. The meaning of each column is as follows.

- **ID test**: The identification of a test case.
- **Observation**: The input value of an observation.
- **Date**: The input value is the date of observation.
- **Expected result**: The output value which is calculated by stakeholders.
- **Aggregate result**: The output value which is calculated by one or several functions of the software program Ontogen.
- **Deducted result**: The output value which is deducted by the SWRLAPI Drools inference engine.

Note that some columns may have sub-columns or maybe disappeared in some tables, depending on the workflow of the considering table.

Table 20 shows the information related to the crop growth workflow. In this workflow, six rules are tested and they have the following identifications: `ADv122019-CG-V7`, `ADv122019-CG-V7d20`, `ADv122019-CG-R1`, `ADv122019-CG-R1d15`, `ADv122019-CG-R5`, and `ADv122019-CG-R5hg45`. They respectively are rules for the **CropGrowth** property of a day to reach one of the states of `V7`, `V7d20`, `R1`, `R1d15`, `R5`, or `R5hg45`. The rules are available in Appendix A. From this table, the expected result is similar to the deducted result. Therefore, it is possible to conclude that all of the rules work correctly.

Table 21 shows the information related to the rain intensity workflow. In this workflow, only one rule is tested. It has the identification: `ADv122019-RI`. This rule is for the **RainIntensity** property of a day to reach one of the states of `Light`,

**Table 20**: Unit tests of rules for the state of CropGrowth deduction

| **ID test** | **Date** (input) | **Observation** (input) | **Expected result** | **Deducted result** |
|---|---|---|---|---|
| CG-V7-25062013000000 | 25/06/2013 00:00:00 | V7 | V7d20 | V7d20 |
| CG-V7-25062013000100 | 25/06/2013 00:01:00 | V7 | V7 | V7 |
| CG-V7-26062013000000 | 26/06/2013 00:00:00 | V7 | V7 | V7 |
| CG-V7-15072013000000 | 15/07/2013 00:00:00 | V7 | V7 | V7 |
| CG-V7-15072013120000 | 15/07/2013 12:00:00 | V7 | V7 | V7 |
| CG-R1-30062013000000 | 30/06/2013 00:00:00 | R1 | R1d15 | R1d15 |
| CG-R1-30062013000100 | 30/06/2013 00:01:00 | R1 | R1 | R1 |
| CG-R1-15072013000000 | 15/07/2013 00:00:00 | R1 | R1 | R1 |
| CG-R1-15072013120000 | 15/07/2013 12:00:00 | R1 | R1 | R1 |
| CG-R5-15072013000000 | 15/07/2013 00:00:00 | R5 | R5 | R5 |
| CG-R5-14072013000000 | 14/07/2013 00:00:00 | R5 | R5 | R5 |
| CG-R5hg45-15072013000000 | 15/07/2013 00:00:00 | R5hg45 | R5hg45 | R5hg45 |
| CG-R5hg45-14072013000000 | 14/07/2013 00:00:00 | R5hg45 | R5hg45 | R5hg45 |

`Moderate`, or `Heavy`. The rule is available in Appendix A. In this table, the expected result column have two sub-columns that contains expected **state** and **delay**. The expected state equals to the deducted state, and expected delay equals to aggregated delay. While the inference engine deducts the state, the function `getDelay(d)` of Ontogen calculates the value of the **DelayDuration** property. It is possible to conclude that the rule works correctly. Note that the function `getDelay(d)` is not the object to be considered in this section since it is not a part of the ontology. However, the delay duration is important for the deduction of crop water need.

Table 22 shows the information related to the soil moisture workflow. In this workflow, only one rule is tested. It has the identification: `ADv122019-RM`. This rule is for the **RootZoneMoistureLevel** property of a day to reach one of the states of `Dry`, `VeryLow`, `Low`, `Average`, `High`, `VeryHigh`, or `Saturated`. The rule is available in Appendix A. The expected result is similar to the deducted result, then it is possible to conclude that the rule works fine.

Table 23 and Table 24 show the information related to the crop water need workflow. In this workflow, 16 rules are tested.

**Table 21**: Unit tests of rules for the state of RainIntensity deduction

| **ID test** | **Observation** (input) | **Expected result** | | **Deducted** | **Aggregated** |
|---|---|---|---|---|---|
| | | **State** | **Delay** | **State** | **Delay** |
| RI-Light-0 | 0 mm | Light | 0 day | Light | 0 day |
| RI-Light-9.9 | 9.9 mm | Light | 0 day | Light | 0 day |
| RI-Moderate-10 | 10 mm | Moderate | 2 day | Moderate | 2 day |
| RI-Moderate-15 | 15 mm | Moderate | 3 day | Moderate | 3 day |
| RI-Moderate-20 | 20 mm | Moderate | 4 day | Moderate | 4 day |
| RI-Moderate-25 | 25 mm | Moderate | 5 day | Moderate | 5 day |
| RI-Moderate-30 | 30 mm | Moderate | 6 day | Moderate | 6 day |
| RI-Moderate-35 | 35 mm | Moderate | 7 day | Moderate | 7 day |
| RI-Moderate-39.9 | 39.9 mm | Moderate | 7 day | Moderate | 7 day |
| RI-Moderate-40 | 40 mm | Heavy | 8 day | Heavy | 8 day |
| RI-Moderate-71.7 | 71.7 mm | Heavy | 14 day | Heavy | 14 day |

Of which, 12 rules are for the **CropWaterNeed** property of a day to reach of the states of `Yes`, `No`, `Unavailable`, or `NotApplicable`. The identification of these rules has radical `ADv122019`, and the code name from `C1` to `C4`. Particularly, the rules with code name `C2` and `C3` are divided into five parts, then it code name add this extra code from `P1` to `P5`. Also, four rules are for calculating the value of the **SleepingDuration** property. The indenticiation codes of these rules are: `ADv122019-SD-P1`, `ADv122019-SD-P2`, `ADv122019-SD-P3`, and `ADv122019-SD-P4`. All of the 16 rules are available in Appendix A.

In the two tables, there are some conventions as follows.

- The duration of the day d is `[17/07/2013 00:00:00, 18/07/2013 00:00:00[`, then the duration of the day d-1 is `[16/07/2013 00:00:00, 17/07/2013 00:00:00[`.
- The observation column has six sub-columns containing four inputs for the day d and two inputs for the day d-1. The expected result and the deducted result have two sub-columns for each.
- Lists of shortcuts and their meanings are:
    - **CWN**: This column contains the state of **CropWaterNeed**.

**Table 22**: Unit tests of rules for the state of RootZoneMoistureLevel deduction

| **ID test** | **Observation** (input) | **Expected result** | **Deducted result** |
|---|---|---|---|
| RZML-Saturated-0 | 0 cbar | Saturated | Saturated |
| RZML-Saturated-9 | 9 cbar | Saturated | Saturated |
| RZML-VeryHigh-10 | 10 cbar | VeryHigh | VeryHigh |
| RZML-VeryHigh-69 | 69 cbar | VeryHigh | VeryHigh |
| RZML-High-70 | 70 cbar | High | High |
| RZML-High-119 | 119 cbar | High | High |
| RZML-Average-120 | 120 cbar | Average | Average |
| RZML-Average-139 | 139 cbar | Average | Average |
| RZML-Low-140 | 140 cbar | Low | Low |
| RZML-Low-149 | 149 cbar | Low | Low |
| RZML-VeryLow-150 | 150 cbar | VeryLow | VeryLow |
| RZML-VeryLow-159 | 159 cbar | VeryLow | VeryLow |
| RZML-Dry-160 | 160 cbar | Dry | Dry |

- **SD**: This column contains the value of **SleepingDuration**.
- **CG**: This column contains the state of **CropGrowth**.
- **RI**: This column contains the state of **RainIntensity**.
- **RZML**: This column contains the state of **RootZoneMoistureLevel**.
- **DD**: This column contains the state of **DelayDuration**.

In most of the test cases, the expected result equals to the deducted result. Except for the test cases from `T1-1` to `T1-8`, all deducted results are `null`. The reason is there is no input of the day d-1, then the corresponding cells are filled with `N/A`. It is possible to conclude that all the rules work correctly.

### 3.6.2 Evaluation Using Sample Test

This research performed a total of 17 sample test scenarios for the four workflows.

- 7 test scenarios for crop growth workflow.

| ID Test | **Observation** (input) | | | | | | **Expected result** | | **Deducted result** | |
|---|---|---|---|---|---|---|---|---|---|---|
| | d-1 | | d [17/07/2013 00:00:00, 18/07/2013 00:00:00[ | | | | | | | |
| | CWN | SD | CG | RI | DD | RZML | CWN | SD | CWN | SD |
| T1-1 | N/A | N/A | V2 | Heavy | 8 | High | NotApplicable | 0 | null | null |
| T1-2 | N/A | N/A | V7 | Heavy | 8 | Average | Yes | 6 | null | null |
| T1-3 | N/A | N/A | V7 | Heavy | 8 | High | Yes | 6 | null | null |
| T1-4 | N/A | N/A | V7 | Heavy | 8 | VeryHigh | No | 0 | null | null |
| T1-5 | N/A | N/A | V7d25 | Heavy | 8 | Average | Yes | 6 | null | null |
| T1-6 | N/A | N/A | V7d25 | Heavy | 8 | High | No | 0 | null | null |
| T1-7 | N/A | N/A | R1 | Heavy | 8 | Low | Yes | 6 | null | null |
| T1-8 | N/A | N/A | R1 | Heavy | 8 | Average | No | 0 | null | null |
| T2-1 | Init | 0 | V7 | Heavy | 8 | High | Yes | 6 | Yes | 6 |
| T2-2 | Init | 0 | V7 | Light | 1 | High | Yes | 6 | Yes | 6 |
| T2-3 | Init | 0 | V7 | Heavy | 8 | VeryHigh | No | 0 | No | 0 |
| T2-4 | Init | 0 | V7d25 | Heavy | 8 | High | No | 0 | No | 0 |
| T2-5 | Init | 0 | V7d25 | Heavy | 8 | Average | Yes | 6 | Yes | 6 |
| T2-6 | Init | 0 | R1 | Heavy | 8 | Average | No | 0 | No | 0 |
| T2-7 | Init | 0 | R1 | Heavy | 8 | Low | Yes | 6 | Yes | 6 |
| T2-8 | Init | 0 | R1d15 | Heavy | 8 | Low | No | 0 | No | 0 |
| T2-9 | Init | 0 | R1d15 | Heavy | 8 | VeryLow | Yes | 6 | Yes | 6 |
| T2-10 | Init | 0 | R5 | Heavy | 8 | Low | No | 0 | No | 0 |
| T2-11 | Init | 0 | R5 | Heavy | 8 | Dry | Yes | 6 | Yes | 6 |
| T3-1 | No | 0 | V7 | Heavy | 8 | VeryHigh | No | 0 | No | 0 |
| T3-2 | No | 0 | V7 | Heavy | 8 | High | Yes | 6 | Yes | 6 |
| T3-3 | No | 0 | V7d25 | Heavy | 8 | High | No | 0 | No | 0 |
| T3-4 | No | 0 | V7d25 | Heavy | 8 | Average | Yes | 6 | Yes | 6 |
| T3-5 | No | 0 | R1 | Heavy | 8 | Average | No | 0 | No | 0 |
| T3-6 | No | 0 | R1 | Heavy | 8 | Low | Yes | 6 | Yes | 6 |
| T3-7 | No | 0 | R1d15 | Heavy | 8 | Low | No | 0 | No | 0 |
| T3-8 | No | 0 | R1d15 | Heavy | 8 | VeryLow | Yes | 6 | Yes | 6 |
| T3-9 | No | 0 | R5 | Heavy | 8 | Low | No | 0 | No | 0 |
| T3-10 | No | 0 | R5 | Heavy | 8 | Dry | Yes | 6 | Yes | 6 |
| T4-1 | Unavailable | 0 | V7 | Light | 1 | VeryHigh | No | 0 | No | 0 |
| T4-2 | Unavailable | 0 | V7 | Light | 1 | High | Yes | 6 | Yes | 6 |
| T4-3 | Unavailable | 0 | V7d25 | Light | 1 | High | No | 0 | No | 0 |
| T4-4 | Unavailable | 0 | V7d25 | Light | 1 | Average | Yes | 6 | Yes | 6 |
| T4-5 | Unavailable | 0 | R1 | Light | 1 | Average | No | 0 | No | 0 |
| T4-6 | Unavailable | 0 | R1 | Light | 1 | Low | Yes | 6 | Yes | 6 |
| T4-7 | Unavailable | 0 | R1d15 | Light | 1 | Low | No | 0 | No | 0 |
| T4-8 | Unavailable | 0 | R1d15 | Light | 1 | VeryLow | Yes | 6 | Yes | 6 |
| T4-9 | Unavailable | 0 | R5 | Light | 1 | Low | No | 0 | No | 0 |
| T4-10 | Unavailable | 0 | R5 | Light | 1 | Dry | Yes | 6 | Yes | 6 |
| T5-1 | Unavailable | 0 | V7 | Moderate | 6 | VeryHigh | No | 0 | No | 0 |
| T5-2 | Unavailable | 0 | V7 | Moderate | 6 | High | Yes | 6 | Yes | 6 |
| T5-3 | Unavailable | 0 | V7d25 | Moderate | 6 | High | No | 0 | No | 0 |
| T5-4 | Unavailable | 0 | V7d25 | Moderate | 6 | Average | Yes | 6 | Yes | 6 |
| T5-5 | Unavailable | 0 | R1 | Moderate | 6 | Average | No | 0 | No | 0 |
| T5-6 | Unavailable | 0 | R1 | Moderate | 6 | Low | Yes | 6 | Yes | 6 |
| T5-7 | Unavailable | 0 | R1d15 | Moderate | 6 | Low | No | 0 | No | 0 |
| T5-8 | Unavailable | 0 | R1d15 | Moderate | 6 | VeryLow | Yes | 6 | Yes | 6 |
| T5-9 | Unavailable | 0 | R5 | Moderate | 6 | Low | No | 0 | No | 0 |
| T5-10 | Unavailable | 0 | R5 | Moderate | 6 | Dry | Yes | 6 | Yes | 6 |

**Table 23**: Unit tests of rules for the state of CropWaterNeed deduction (part 1)

- 3 test scenarios for rain intensity workflow.
- 7 test scenarios for soil moisture workflow.

| ID Test | Observation (input) | | | | | | Expected result | | Deducted result | |
|---|---|---|---|---|---|---|---|---|---|---|
| | d-1 | | d [17/07/2013 00:00:00, 18/07/2013 00:00:00[ | | | | | | | |
| | CWN | SD | CG | RI | DD | RZML | CWN | SD | CWN | SD |
| T6-1 | Unavailable | 0 | V7 | Heavy | 8 | VeryHigh | No | 0 | No | 0 |
| T6-2 | Unavailable | 0 | V7 | Heavy | 8 | High | Yes | 6 | Yes | 6 |
| T6-3 | Unavailable | 0 | V7d25 | Heavy | 8 | High | No | 0 | No | 0 |
| T6-4 | Unavailable | 0 | V7d25 | Heavy | 8 | Average | Yes | 6 | Yes | 6 |
| T6-5 | Unavailable | 0 | R1 | Heavy | 8 | Average | No | 0 | No | 0 |
| T6-6 | Unavailable | 0 | R1 | Heavy | 8 | Low | Yes | 6 | Yes | 6 |
| T6-7 | Unavailable | 0 | R1d15 | Heavy | 8 | Low | No | 0 | No | 0 |
| T6-8 | Unavailable | 0 | R1d15 | Heavy | 8 | VeryLow | Yes | 6 | Yes | 6 |
| T6-9 | Unavailable | 0 | R5 | Heavy | 8 | Low | No | 0 | No | 0 |
| T6-10 | Unavailable | 0 | R5 | Heavy | 8 | Dry | Yes | 6 | Yes | 6 |
| T7-1 | Unavailable | 1 | V7 | Moderate | 6 | VeryHigh | Unavailable | 6 | Unavailable | 6 |
| T7-2 | Unavailable | 1 | V7 | Moderate | 6 | High | Unavailable | 6 | Unavailable | 6 |
| T7-3 | Unavailable | 1 | V7d25 | Moderate | 6 | High | Unavailable | 6 | Unavailable | 6 |
| T7-4 | Unavailable | 1 | V7d25 | Moderate | 6 | Average | Unavailable | 6 | Unavailable | 6 |
| T7-5 | Unavailable | 1 | R1 | Moderate | 6 | Average | Unavailable | 6 | Unavailable | 6 |
| T7-6 | Unavailable | 1 | R1 | Moderate | 6 | Low | Unavailable | 6 | Unavailable | 6 |
| T7-7 | Unavailable | 1 | R1d15 | Moderate | 6 | Low | Unavailable | 6 | Unavailable | 6 |
| T7-8 | Unavailable | 1 | R1d15 | Moderate | 6 | VeryLow | Unavailable | 6 | Unavailable | 6 |
| T7-9 | Unavailable | 1 | R5 | Moderate | 6 | Low | Unavailable | 6 | Unavailable | 6 |
| T7-10 | Unavailable | 1 | R5 | Moderate | 6 | Dry | Unavailable | 6 | Unavailable | 6 |
| T8-1 | Unavailable | 1 | V7 | Heavy | 8 | VeryHigh | Unavailable | 8 | Unavailable | 8 |
| T8-2 | Unavailable | 1 | V7 | Heavy | 8 | High | Unavailable | 8 | Unavailable | 8 |
| T8-3 | Unavailable | 1 | V7d25 | Heavy | 8 | High | Unavailable | 8 | Unavailable | 8 |
| T8-4 | Unavailable | 1 | V7d25 | Heavy | 8 | Average | Unavailable | 8 | Unavailable | 8 |
| T8-5 | Unavailable | 1 | R1 | Heavy | 8 | Average | Unavailable | 8 | Unavailable | 8 |
| T8-6 | Unavailable | 1 | R1 | Heavy | 8 | Low | Unavailable | 8 | Unavailable | 8 |
| T8-7 | Unavailable | 1 | R1d15 | Heavy | 8 | Low | Unavailable | 8 | Unavailable | 8 |
| T8-8 | Unavailable | 1 | R1d15 | Heavy | 8 | VeryLow | Unavailable | 8 | Unavailable | 8 |
| T8-9 | Unavailable | 1 | R5 | Heavy | 8 | Low | Unavailable | 8 | Unavailable | 8 |
| T8-10 | Unavailable | 1 | R5 | Heavy | 8 | Dry | Unavailable | 8 | Unavailable | 8 |
| T9-1 | Unavailable | 2 | V7 | Heavy | 8 | VeryHigh | Unavailable | 9 | Unavailable | 9 |
| T9-2 | Unavailable | 2 | V7 | Moderate | 6 | VeryHigh | Unavailable | 7 | Unavailable | 7 |
| T9-3 | Unavailable | 2 | V7 | Moderate | 2 | VeryHigh | Unavailable | 3 | Unavailable | 3 |
| T9-4 | Unavailable | 2 | V7 | Light | 1 | VeryHigh | Unavailable | 2 | Unavailable | 2 |

**Table 24**: Unit tests of rules for the state of CropWaterNeed deduction (part 2)

In each test scenario, the system takes one input value, which is the date of the observation in the Arvalis dataset. From the date value, the system can find the other corresponding observations in the Arvalis dataset. The SWRLAPI Drools inference engine applies each rule to the knowledge base, including the input value, to infer a deducted result. It is possible to evaluate the rule by comparing the deducted result with the expected result calculated by stakeholders. The information related to the comparisons is available in the different tables corresponding to workflows. Each table contains several columns. The meaning of each column is as follows.

- **ID test**: The identification of a test case.
- **Date**: The input value is the date of observation.
- **Expected result**: The output value which is calculated by stakeholders.

- **Deducted result**: The output value which is deducted by the SWRLAPI Drools inference engine.

Table 25 shows the information related to the crop growth workflow. Similar to Section 3.6.1, this workflow has six rules to be tested: `ADv122019-CG-V7`, `ADv122019-CG-V7d20`, `ADv122019-CG-R1`, `ADv122019-CG-R1d15`, `ADv122019-CG-R5`, and `ADv122019-CG-R5hg45`. They respectively are rules for the **CropGrowth** property of a day to reach one of the states of `V7`, `V7d20`, `R1`, `R1d15`, `R5`, or `R5hg45`. The rules are available in Appendix A. In this table, the expected result equals to the deducted result. It is possible to conclude that all of the rules work correctly.

**Table 25**: Sample tests of rules for the state of CropGrowth deduction

| **ID test** | **Date** (input) | **Expected result** | **Deducted result** |
|---|---|---|---|
| CG-V7-26062013 | 26/06/2013 | V7 | V7 |
| CG-V7d20-16072013 | 16/07/2013 | V7d20 | V7d20 |
| CG-R1-27072013 | 27/07/2013 | R1 | R1 |
| CG-R1-31072013 | 31/07/2013 | R1 | R1 |
| CG-R1d15-11082013 | 11/08/2013 | R1d15 | R1d15 |
| CG-R5-10092013 | 10/09/2013 | R5 | R5 |
| CG-R5hg45-19092013 | 19/09/2013 | R5hg45 | R5hg45 |

Table 26 shows the information related to the rain intensity workflow. Similar to Section 3.6.1, this workflow has one rule to be tested: `ADv122019-RI`. This rule is for the **RainIntensity** property of a day to reach one of the states of `Light`, `Moderate`, or `Heavy`. The rule is available in Appendix A. In this table, the expected result equals to the deducted result. Therefore, it is possible to conclude that the rule works correctly.

**Table 26**: Sample tests of rules for the state of RainIntensity deduction

| **ID test** | **Date** (input) | **Expected result** | **Deducted result** |
|---|---|---|---|
| RI-Light-16072013 | 16/07/2013 | Light | Light |
| RI-Moderate-28072013 | 28/07/2013 | Moderate | Moderate |
| RI-Heavy-31072013 | 31/07/2013 | Heavy | Heavy |

Table 27 shows the information related to the soil moisture workflow. Similar to Section 3.6.1, this workflow has one rule to be tested: `ADv122019-RM`. This rule is for the **RootZoneMoistureLevel** property of a day to reach one of the states of `Dry`, `VeryLow`, `Low`, `Average`, `High`, `VeryHigh`, or `Saturated`. The rule is available in Appendix A. The expected result is similar to the deducted result; then, it is possible to conclude that the rule works fine.

**Table 27**: Sample tests of rules for the state of RootZoneMoistureLevel deduction

| **ID test** | **Date** (input) | **Expected result** | **Deducted result** |
|---|---|---|---|
| RZML-Saturated-26062013 | 26/06/2013 | Saturated | Saturated |
| RZML-VeryHigh-16072013 | 16/07/2013 | VeryHigh | VeryHigh |
| RZML-Low-21072013 | 21/07/2013 | Low | Low |
| RZML-High-26072013 | 26/07/2013 | High | High |
| RZML-VeryLow-31072013 | 31/07/2013 | VeryLow | VeryLow |
| RZML-Average-03082013 | 03/08/2013 | Average | Average |
| RZML-Dry-14082013 | 14/08/2013 | Dry | Dry |

### 3.6.3 Evaluation Using System Test

The system test is to run the DSS over data in a period of the original dataset. Those periods should have different events that imply state changes in the observed properties. The four selected periods are as follows.

- `[14/08/2013, 21/08/2013]`: The period contains one light rain event.
- `[21/08/2013, 03/09/2013]`: The period contains one moderate rain event.
- `[31/07/2013, 14/08/2013]`: The period contains two consecutive moderate rain events.
- `[22/07/2013, 31/07/2013]`: The period contains one crop state change and two rain events.

Each period contains two consecutive watering days which are the first day and the last day in the period. For example, in the first period `[14/08/2013, 21/08/2013]`, the watering days are 14/08/2013 and 21/08/2013.

Then, this test compares the day of the watering decision made by the DSS from the day of the watering decision made by a farmer. The comparison relies on the fact that when both the farmer and the DSS follow the same rules of the IRRINOV®

method, then the watering decisions on the same conditions should be equivalent. Thus, when both the farmer and the DSS decide to do a watering on the first day of each period, the next watering day deduced by the DSS should be the last day of the period, in line with the decision of the farmer.

Note that with each period, the DSS takes the human decision of the first day as an input. It produces its decision from the second day of the period. For example, with the period `[14/08/2013, 21/08/2013]`, the DSS takes the watering decision on day 14/08/2013. It deduces the decisions from day 15/08/2013 to day 21/08/2013.

The results of the experiment over the four mentioned periods are shown in the four data tables below. Each table includes 16 columns and several rows. The first row of the table contains the labels of the type of data in the column. Each one from the second row contains the data corresponding to a day of the period. Details of the first row: The first 13 cells are filled in gray to indicate that the data in the columns corresponding to these cells are input data; the $14^{\text{th}}$ and $15^{\text{th}}$ cells are in red to indicate that the data in the columns corresponding to these cells are inferred results. The last cell is in green to indicate that the data in the 16th column are the decisions of humans in reality. The meaning of each column is as follows.

- The first three columns show the information of the day, month, and year.
- The fourth column shows the total rain quantity of a day. Its unit is mm. When the rain quantity is greater than 0, the cell is filled with blue.
- The fifth column shows the number of delay days. These data in this cell are calculated from the data in the fourth column. Its unit is the day. When the number of delay days is greater than 0, the cell is filled with blue.
- The next six cells correspond to six soil tensiometers. These indicate the average soil moisture of a day as measured by a soil tensiometer. Its unit is cbar.
- The $12^{\text{th}}$ cell shows the average root zone moisture of a day. The data in this cell are calculated based on the data in the six previous cells. Its unit is cbar.
- The $13^{\text{th}}$ cell shows the crop state observed by a human. If farmers recognize a new crop state and put it in the dataset, then this state is written by the name code, and the cell is filled with yellow. Otherwise, the text in the cell is `null`.
- The $14^{\text{th}}$ cell shows the value of the **SleepingDuration** property inferred by the DSS. Its unit is the day.
- The $15^{\text{th}}$ cell shows the state of the **CropWaterNeed** property. When **CropWaterNeed** reaches state `Yes`, the text in the cell is in uppercase, and the cell is filled with red.
- The $16^{\text{th}}$ cell shows the decision of farmers recorded in the dataset. The day that farmers decide to water the plot, the corresponding cell is made green, and

the text inside the cell is `YES`. Otherwise, there is no text.

Table 28 shows the results of the experimental period from 22/07/2013 to 31/07/2013. The first watering is performed on 22/07/2013. The farmer decides to schedule the second watering on 31/07/2013; however, the DSS suggests that the farmer does nothing on the same day. The farmer disobeys the method IRRINOV® in this case for a good reason. When the value of the average root zone moisture is 152 cbar, the root zone moisture level is low. If the root zone moisture level is low and the crop growth is R1, then the crop needs water. If following the IRRINOV® methods, the farmer should wait at least two more days because a rain event postpones the irrigation by four days. The data of SleepingDuration also reflect this calculation. The result of this experiment proves that the farmer decision may be different from the method IRRINOV®.

| Day | Month | Year | Rain | DelayDuration | WM-01 | WM-02 | WM-03 | WM-04 | WM-05 | WM-06 | RzM | CropState | SleepingDuration | Ontogen-CWN | Arvalis-CWN |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 22 | 7 | 2013 | 0 | 0 | 135 | 111 | 81 | 106 | 22 | 60 | 171 | null | 6 | YES | YES |
| 23 | 7 | 2013 | 6 | 0 | 70 | 31 | 1 | 18 | 0 | 0 | 31 | null | 5 | unavailable | |
| 24 | 7 | 2013 | 0 | 0 | 84 | 50 | 3 | 44 | 0 | 0 | 50 | null | 4 | unavailable | |
| 25 | 7 | 2013 | 0 | 0 | 92 | 56 | 12 | 112 | 12 | 0 | 68 | null | 3 | unavailable | |
| 26 | 7 | 2013 | 0 | 0 | 107 | 67 | 38 | 144 | 28 | 36 | 103 | null | 2 | unavailable | |
| 27 | 7 | 2013 | 0 | 0 | 126 | 83 | 64 | 158 | 61 | 69 | 152 | R1 | 1 | unavailable | |
| 28 | 7 | 2013 | 20 | 4 | 141 | 67 | 4 | 43 | 3 | 0 | 70 | null | 4 | unavailable | |
| 29 | 7 | 2013 | 0 | 0 | 144 | 89 | 16 | 112 | 10 | 1 | 99 | null | 3 | unavailable | |
| 30 | 7 | 2013 | 0 | 0 | 147 | 97 | 29 | 150 | 19 | 10 | 116 | null | 2 | unavailable | |
| 31 | 7 | 2013 | 0 | 0 | 149 | 101 | 46 | 169 | 47 | 51 | 152 | null | 1 | unavailable | YES |

**Table 28**: Experiment during the period from 22/07/2013 to 31/07/2013.

Table 29 shows the results of the experimental period from 31/07/2013 to 14/08/2013. The first watering is performed on 31/07/2013. The second watering is scheduled by both the farmer and the DSS is on 31/07/2013. This example shows that the farmer follows the IRRINOV® method and the DSS gives a correct result based on this method. The DSS result is correct in this case because there are rains on two consecutive days. The two rainy days are 06/08/2013 and 07/08/2013. These two rainy days produce seven delay days.

Table 30 shows the results of the experimental period from 14/08/2013 to 21/08/2013. The first watering is performed on 14/08/2013. The second watering is scheduled by both the farmer and the DSS is on 21/08/2013. This example shows that the farmer follows the method IRRINOV® and the DSS gives a correct result based on this method. The DSS result is correct in this case because of the light rain on 19/08/2013. There are no delay days in this period.

Table 31 shows the results of the experimental period from 21/08/2013 to 03/09/2013. The first watering is performed on 21/08/2013. The farmer decides to schedule the second watering on 21/08/2013. However, the second watering is

| Day | Month | Year | Rain | DelayDuration | WM-01 | WM-02 | WM-03 | WM-04 | WM-05 | WM-06 | RzM | CropState | SleepingDuration | Ontogen-CWN | Arvalis-CWN |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | 7 | 2013 | 0 | 0 | 149 | 101 | 46 | 169 | 47 | 51 | 152 | null | 6 | YES | YES |
| 1 | 8 | 2013 | 0 | 0 | 50 | 0 | 1 | 8 | 16 | 0 | 9 | null | 5 | unavailable | |
| 2 | 8 | 2013 | 0 | 0 | 75 | 0 | 10 | 90 | 54 | 0 | 64 | null | 4 | unavailable | |
| 3 | 8 | 2013 | 0 | 0 | 91 | 19 | 37 | 147 | 98 | 0 | 135 | null | 3 | unavailable | |
| 4 | 8 | 2013 | 0 | 0 | 103 | 29 | 56 | 162 | 129 | 1 | 185 | null | 2 | unavailable | |
| 5 | 8 | 2013 | 0 | 0 | 116 | 41 | 78 | 167 | 137 | 27 | 215 | null | 1 | unavailable | |
| 6 | 8 | 2013 | 27 | 5 | 133 | 59 | 104 | 185 | 151 | 88 | 255 | null | 5 | unavailable | |
| 7 | 8 | 2013 | 11 | 2 | 136 | 0 | 2 | 20 | 11 | 0 | 13 | null | 6 | unavailable | |
| 8 | 8 | 2013 | 0 | 0 | 123 | 0 | 0 | 69 | 5 | 0 | 5 | null | 5 | unavailable | |
| 9 | 8 | 2013 | 0 | 0 | 90 | 0 | 0 | 160 | 14 | 0 | 14 | null | 4 | unavailable | |
| 10 | 8 | 2013 | 0 | 0 | 75 | 0 | 0 | 189 | 32 | 0 | 32 | null | 3 | unavailable | |
| 11 | 8 | 2013 | 0 | 0 | 71 | 9 | 14 | 203 | 81 | 0 | 95 | null | 2 | unavailable | |
| 12 | 8 | 2013 | 0 | 0 | 69 | 19 | 26 | 199 | 128 | 0 | 154 | null | 1 | unavailable | |
| 13 | 8 | 2013 | 0 | 0 | 72 | 28 | 42 | 201 | 156 | 0 | 198 | null | 0 | unavailable | |
| 14 | 8 | 2013 | 0 | 0 | 78 | 39 | 59 | 199 | 177 | 4 | 236 | null | 6 | YES | YES |

**Table 29**: Experiment during the period from 31/07/2013 to 14/08/2013.

| Day | Month | Year | Rain | DelayDuration | WM-01 | WM-02 | WM-03 | WM-04 | WM-05 | WM-06 | RzM | CropState | SleepingDuration | Ontogen-CWN | Arvalis-CWN |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 14 | 8 | 2013 | 0 | 0 | 78 | 39 | 59 | 199 | 177 | 4 | 236 | null | 6 | YES | YES |
| 15 | 8 | 2013 | 0 | 0 | 36 | 1 | 1 | 31 | 15 | 0 | 16 | null | 5 | unavailable | |
| 16 | 8 | 2013 | 0 | 0 | 43 | 0 | 0 | 136 | 72 | 0 | 72 | null | 4 | unavailable | |
| 17 | 8 | 2013 | 0 | 0 | 47 | 1 | 16 | 207 | 154 | 0 | 170 | null | 3 | unavailable | |
| 18 | 8 | 2013 | 0 | 0 | 49 | 16 | 27 | 210 | 174 | 0 | 201 | null | 2 | unavailable | |
| 19 | 8 | 2013 | 2 | 0 | 54 | 26 | 43 | 211 | 185 | 0 | 228 | null | 1 | unavailable | |
| 20 | 8 | 2013 | 0 | 0 | 56 | 31 | 45 | 207 | 190 | 1 | 235 | null | 0 | unavailable | |
| 21 | 8 | 2013 | 0 | 0 | 60 | 40 | 59 | 188 | 220 | 19 | 247 | null | 6 | YES | YES |

**Table 30**: Experiment during the period from 14/08/2013 to 21/08/2013.

scheduled by the DSS is two days sooner. From this experiment, it is possible to conclude that the farmer disobeys the method IRRINOV® one more time. However, this time, the reason for the farmer is unclear. From the data in the column of average root zone moisture, the root zone is at a dry state from 29/08/2013. Therefore, the decision following the method IRRINOV® is better than the decision made by the farmer, since the crops in the latter case need to wait for water more than those in the former case.

From the result of system tests and the discussion of stakeholders on this result, it is possible to conclude that the rules of IRRIG work correctly. The difference between the watering decisions recorded in the Arvalis dataset and the watering decision suggested by the DSS is due to the limit of IRRINOV®, and the errors made during the Arvalis experimentation.

## 3.7 Summary and Discussion

To sum up, this chapter presents the procedure to develop an irrigation DSS. This procedure relies on a new system development methodology, which is the combination

| Day | Month | Year | Rain | DelayDuration | WM-01 | WM-02 | WM-03 | WM-04 | WM-05 | WM-06 | RzM | CropState | SleepingDuration | Ontogen-CWN | Arvalis-CWN |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 21 | 8 | 2013 | 0 | 0 | 60 | 40 | 59 | 188 | 220 | 19 | 247 | null | 6 | YES | YES |
| 22 | 8 | 2013 | 0 | 0 | 24 | 2 | 1 | 28 | 109 | 0 | 30 | null | 5 | unavailable | |
| 23 | 8 | 2013 | 0 | 0 | 36 | 0 | 3 | 147 | 188 | 0 | 150 | null | 4 | unavailable | |
| 24 | 8 | 2013 | 0 | 0 | 40 | 5 | 19 | 199 | 203 | 0 | 218 | null | 3 | unavailable | |
| 25 | 8 | 2013 | 22 | 4 | 42 | 16 | 25 | 212 | 200 | 0 | 225 | null | 6 | unavailable | |
| 26 | 8 | 2013 | 4.5 | 0 | 45 | 10 | 3 | 51 | 142 | 0 | 61 | null | 5 | unavailable | |
| 27 | 8 | 2013 | 0 | 0 | 1 | 2 | 18 | 23 | 0 | 255 | 25 | null | 4 | unavailable | |
| 28 | 8 | 2013 | 0 | 0 | 0 | 1 | 51 | 89 | 0 | 255 | 90 | null | 3 | unavailable | |
| 29 | 8 | 2013 | 0 | 0 | 0 | 1 | 132 | 172 | 0 | 255 | 173 | null | 2 | unavailable | |
| 30 | 8 | 2013 | 0 | 0 | 3 | 9 | 182 | 202 | 0 | 255 | 211 | null | 1 | unavailable | |
| 31 | 8 | 2013 | 0 | 0 | 15 | 18 | 209 | 239 | 0 | 255 | 257 | null | 0 | unavailable | |
| 1 | 9 | 2013 | 0 | 0 | 24 | 29 | 229 | 239 | 0 | 255 | 268 | null | 6 | YES | |
| 2 | 9 | 2013 | 0 | 0 | 39 | 56 | 239 | 239 | 46 | 255 | 239 | null | 5 | unavailable | |
| 3 | 9 | 2013 | 0 | 0 | 0 | 3 | 33 | 36 | 0 | 255 | 36 | null | 4 | unavailable | YES |

**Table 31**: Experiment during the period from 21/08/2013 to 03/09/2013.

of LOT and Mini-waterfall. LOT is the methodology to develop IRRIG, the ontology to model the system's data. Mini-waterfall is the methodology to develop the software program of the DSS. This development procedure includes four steps. First, in the specification step, system developers analyze all sources of information related to the system, then produce a specification and a competency questions documents. Second, from the specification and competency questions documents, system developers can form an ontology specialized for this case study. Moreover, they can determine the services run by this DSS. Third, the implementation step is to write an algorithm to run the system and to code rules for the reasoning. The fourth step is to evaluate the ontology IRRIG and the system using several testing types.

It is possible to conclude that this chapter success in building an irrigation DSS and an ontology specialized for this case study. However, there are still two points to discuss further:

- In addition to deducting the crop growth state based on human observation, another approach is to use the Growing Degree Unit (GDU) method. This method proposes to determine the maximum and minimum temperature of a day to estimate the heat accumulate of crops. This value also enables the deducting of the crop growth state *(Lori J. Abendroth et al., 2011)*. Even that IRRIG provides enough vocabulary to describe the temperature data for calculating GDU, the rules relating to this method are not yet defined. One future version of IRRIG should provide these rules coded in SWRL.
- IRRIG should define vocabulary specialized for agriculture, which is missing in available vocabularies, such as the unit `millibar`.

# Conclusion

*"Life is like riding a bicycle. To keep your balance you must keep moving."* – **Albert Einstein**

---

The emergence of the IoT, together with the new wave of ICTs, brings opportunities and challenges in e-agriculture. In particular, they promote this domain to a new revolution called Agriculture 4.0. In Agriculture 4.0, farmers normally use a CAS to manage and control agricultural activities because it can react fast and adequately to the context changes. The data generated from a CAS should be sharable not only in e-agriculture but also in other domains such as smart cities and smart environments. The combination of them creates a cross-domain ecosystem of the IoT. Besides these advantages, using CAS in e-agriculture encounters some challenges specialized in agriculture. First, system developers would need to upgrade their CAS frequently: agriculture is a seasonal activity, and its workplace environment is outdoors, so it implies many unpredictable factors that demand the changes in software and hardware of an agriculture CAS. Consequently, there is a need to upgrade the system using new hardware and software without changing the CAS's functionality. Address the first challenge; this research proposes a stack of services for CASs, which allows designing a system by services. This approach focuses on the goal of services and reduces the system's dependency on hardware and software. Thus, it improves the upgradability of CASs. Second, the heterogeneity data generated from a CASs is a challenge in the communication between the components in one CAS or even more, the data sharing between different CASs. Address the second challenge; this research proposes CASO, an ontology dedicated to CASs. It provides a vocabulary to model the computations available in CASs and data generated from these computations. Also, this ontology implies a mechanism to create rules for reasoning. A CAS using CASO can improve the interoperability of data semantic between its devices. Moreover, when different CASs use this ontology, the interoperability of data semantic between such systems is improved. The third contribution of this research is an irrigation DSS. It is a part of the irrigation CAS in TSCF. It automizes an irrigation method called IRRINOV®.

The two first contributions of this project are used in building this DSS. Indeed, this DSS is designed using the stack of services for CASs and it runs six services: **annotation**, **storage**, **retrieving**, **merging**, **reasoning**, **storage updating**. It also uses CASO to develop a specialized ontology for the case study, named IRRIG. The table decisions in the IRRINOV® are transformed into the rules for reasoning and are a part of IRRIG.

The contributions of this research are from the real issues in e-agriculture. Also, the data resources used to evaluate them are from several works in agriculture. However, the two first contributions of this thesis are generalized to be generic. It means that they can be applied in other domains in the IoT ecosystem, such as smart cities and smart transportation.

On the practical side, the DSS developed in this research contributes to an irrigation CAS in TSCF. The system developers of this project deploy the other parts of the CAS in TechnoHall[10], INRAE. Figure 57 illustrates the deployment of the devices of the CAS irrigation system in TechnoHall. Moreover, as in the plan of TSCF, the irrigation CAS irrigation system will further employ in the AgrotechnoPôle. Figure 57 shows a picture of the AgrotechnoPôle.

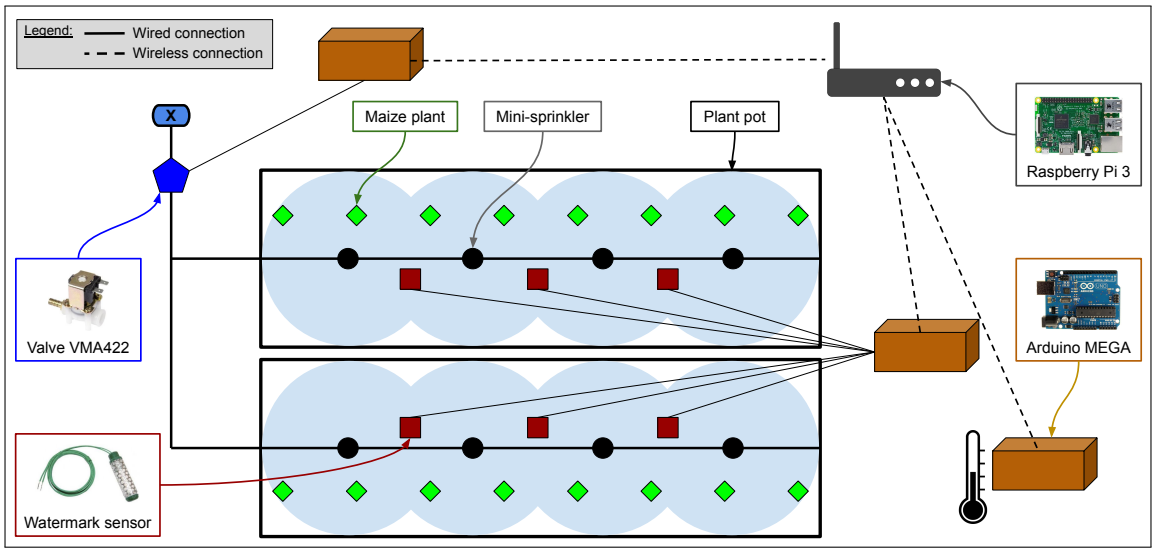



**Figure 57**: Devices of the CAS irrigation system in TechnoHall

There are several perspectives developed from this research. The following four are the most appealing and executable.

- It is possible to develop a vocabulary to describe the networking of CASs. It could be a part of CASO since networking is an aspect of a CAS. Moreover,

[10]TechnoHall is a basement for technologies experimentations located in the building of the center INRAE Clermont-Auvergne-Rhône-Alpes. TechnoHall is under the supervision of TSCF.

**Figure 58**: AgroTechnoPôle in Montoldre, France

with this networking part, it is possible to use reasoning techniques to force a node to make a decision automatically. An example of multihoming: there are two gateways in a local WSN. A node can send data to an external network via one of them. The information about the two gateways and the network could be useful for a node to select the path that optimizes the network traffic. This perspective requires an expert in networking that can analyze and provide a list of requirements. This expert may need to translate networking algorithms into rules. The challenge of this perspective is to create an expert system that is small enough to put in every node in the local network.

- A CAS could be smarter if it becomes an ACAS. The adaptive feature allows a CAS to change its configurations based on the context. It is necessary to back to the example of the irrigation CAS in TSCF to prove this statement. Given that a probe sends soil tension information to the server once per day. When the soil tension is near the threshold that triggers the watering activity, the server is unable to make this decision in this day, but need to wait to the next day. An ACAS can change the communication frequency between the probe and the server from 24 hours to 1 hour for that special day. Thus, the decision is more on time. This perspective requires an ontological expert that can transform such constraints into rules. Also, it needs a vocabulary to describe the systems' configuration information, such as the communication frequency.
- Each service in the stack of services for CASs should follow the principle of the microservice design mindset: to have two parts. One part is to execute

the goal. Another one handles the communication between services. For example, the service **reasoning** has two parts. While the first part performs the reasoning, the second one has the responsibility to receives data from the service **retrieving** or send data to the service **context transformation**. This perspective requires a system engineer who can operate a software program for the first part and be able to build a communication protocol for the second part. One suggestion is to use MQTT for the second part since this protocol follows the publish/subscribe mechanism, and it is easy to implement.

- The irrigation method IRRINOV® cannot cover all situations; then, the DSS sometimes might produce fault suggestion. One solution to overcome this inconvenience is to use machine learning technique. When the input data is large enough, and the output based on the decisions of agronomists are all correct, this technique can produce the new rules into the rules base. This perspective requires an expert in machine learning and an agronomist. The input data could be real experiment data, or a software program could generate them randomly. The key is that the agronomist must provide the answer from the input data. The machine learning expert has knowledge of rules extraction from training data.

# References


Aberer, K., Hauswirth, M., and Salehi, A. (2006). A Middleware for Fast and Flexible Sensor Network Deployment. In *Proceedings of the 32nd International Conference on Very Large Data Bases*, VLDB '06, pages 1199–1202, Seoul, Korea.

Abowd, G. D., Dey, A. K., Brown, P. J., Davies, N., Smith, M., and Steggles, P. (1999). Towards a Better Understanding of Context and Context-Awareness. In *Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing, HUC '99*, pages 304–307, Karlsruhe, Germany. Springer-Verlag London, UK.

Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., and Ayyash, M. (2015). Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Communications Surveys & Tutorials*, 17(4):2347–2376.

Alobaid, A., Garijo, D., Poveda-Villalón, M., Santana-Perez, I., Fernández-Izquierdo, A., and Corcho, O. (2019). Automating ontology engineering support activities with OnToology. *Journal of Web Semantics*, 57:100472.

Arvalis, INRA, and Chambre d'Agriculture (2007). Guide de l'utilisateur, Carnet de terrain: Piloter l'irrigation avec la méthode IRRINOV. Technical report 07X04, Arvalis, Midy-Pyrénées, France.

Bajceta, M., Sekulic, P., Krstajic, B., Djukanovic, S., and Popovic, T. (2016). A private IoT cloud platform for precision agriculture and ecological monitoring. In *International Conference on Electrical, Electronic and Computing Engineering*, pages 1–6, Zlatibor, Serbia.

Bechhofer, S. and Miles, A. (2009). Skos simple knowledge organization system reference. *W3C recommendation, W3C*.

Bendadouche, R., Roussey, C., De Sousa, G., Chanet, J., and Hou, K. (2012). Extension of the Semantic Sensor Network Ontology for Wireless Sensor Networks: The Stimulus-WSNnode-Communication Pattern. In *5th International Workshop*

*on Semantic Sensor Networks in conjunction with the 11th International Semantic Web Conference (ISWC)*, pages 1–16, Boston, United States.

Berners-Lee, T., Hendler, J., and Lassila, O. (2001). The semantic Web. *Scientific American*, 284(5):34–43.

Brouwer, C., Prins, K., and Heibloem, M. (1989). Irrigation water management: irrigation scheduling. *FAO training manual*, 4:66.

Buratti, C., Conti, A., Dardari, D., and Verdone, R. (2009). An Overview on Wireless Sensor Networks Technology and Evolution. *Sensors*, 9(9):6869–6896.

Calbimonte, J.-P., Sarni, S., Eberle, J., and Aberer, K. (2014). XGSN: An Open-source Semantic Sensing Middleware for the Web of Things. In *TC/SSN@ ISWC*, pages 51–66, Riva del Garda, Trentino, Italy.

Compton, M., Barnaghi, P., Bermudez, L., GarcíA-Castro, R., Corcho, O., Cox, S., Graybeal, J., Hauswirth, M., Henson, C., and Herzog, A. (2012a). The SSN ontology of the W3c semantic sensor network incubator group. *Web semantics: science, services and agents on the World Wide Web*, 17:25–32.

Compton, M., Barnaghi, P., Bermudez, L., García-Castro, R., Corcho, O., Cox, S., Graybeal, J., Hauswirth, M., Henson, C., Herzog, A., Huang, V., Janowicz, K., Kelsey, W. D., Le Phuoc, D., Lefort, L., Leggieri, M., Neuhaus, H., Nikolov, A., Page, K., Passant, A., Sheth, A., and Taylor, K. (2012b). The SSN ontology of the W3c semantic sensor network incubator group. *Web Semantics: Science, Services and Agents on the World Wide Web*, 17:25–32.

Deepika, G. and Rajapirian, P. (2016). Wireless sensor network in precision agriculture: A survey. In *2016 International Conference on Emerging Trends in Engineering, Technology and Science (ICETETS)*, pages 1–4, Pudukkottai, India. IEEE.

Direction générale de la modernisation de l'État (DGME) (2009). Reference General d'Interoperability RGI version 1.0.

Efstratiou, C. (2004). *Coordinated Adaptation for Adaptive Context-Aware Applications*. Phd thesis, Lancaster University, Lancashire, England.

ETSI (2015). ETSI TS 264 411 - v1.1.1. SmartM2M; Smart Appliances; Reference Ontology and oneM2M Mapping. Technical report, ETSI.

ETSI TS 103 264 - v2.1.1 (2017). SmartM2M; Smart Appliances; Reference Ontology and oneM2M Mapping. Technical report, ETSI.

Gaire, R., Lefort, L., Compton, M., Falzon, G., Lamb, D. W., and Taylor, K. (2013). Semantic web enabled smart farming. In *Proceedings of the 1st International Workshop on Semantic Machine Learning and Linked Open Data (SML2OD2013) for Agricultural and Environmental Informatics*, pages 1–6, Sydney, Australia.

García-Castro, R., Fernández-Izquierdo, A., Heinz, C., Kostelnik, P., Poveda-Villalón, M., and Serena, F. (2017). D2.2 Detailed Specification of the Semantic Model. Technical report, Universidad Politécnica de Madrid (UPM).

Garijo, D. (2017). Widoco: a wizard for documenting ontologies. In *International Semantic Web Conference*, pages 94–102. Springer.

Garijo, D. and Poveda Villalon, M. (2017). A checklist for complete vocabulary metadata. Technical report, WIDOCO.

Goumopoulos, C. (2012). An Autonomous Wireless Sensor/Actuator Network for Precision Irrigation in Greenhouses. In *Smart Sensing Technology for Agriculture and Environmental Monitoring*, pages 1–20. Springer, Berlin, Heidelberg.

Goumopoulos, C., Kameas, A. D., and Cassells, A. (2009). An Ontology-Driven System Architecture for Precision Agriculture Applications. *Int. J. Metadata Semant. Ontologies*, 4(1/2):72–84.

Goumopoulos, C., O'Flynn, B., and Kameas, A. (2014). Automated zone-specific irrigation with wireless sensor/actuator network and adaptable decision support. *Computers and Electronics in Agriculture*, 105:20–33.

Griffith, C., Heydon, G., Lamb, D., Lefort, L., Taylor, K., and Trotter, M. (2013). Smart Farming: Leveraging the impact of broadband and the digital economy. Technical report, CSIRO and University of New England, Australia.

Gyrard, A., Bonnet, C., Boudaoud, K., and Serrano, M. (2016). Lov4iot: A second life for ontology-based domain knowledge to build semantic web of things applications. In *IEEE 4th International Conference on Future Internet of Things and Cloud (FiCloud)*, pages 254–261, Vienna, Austria. IEEE.

Gómez-Pérez, A., Fernandez-Lopez, M., and Corcho, O. (2004). *Ontological Engineering: with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web.* Advanced Information and Knowledge Processing. Springer-Verlag, London, 1st edition.

Harari, Y. N. (2015). *Sapiens: a brief history of humankind.* Popular science. Vintage Books, London, 1st edition.

Hodgson, R., Keller, P. J., Hodges, J., and Spivak, J. (2014). Qudt-quantities, units, dimensions and data types ontologies.

Hwang, J., Shin, C., and Yoe, H. (2010). A Wireless Sensor Network-Based Ubiquitous Paprika Growth Management System. *Sensors*, 10(12):11566–11589.

Hwang, J. and Yoe, H. (2011). Study on the Context-Aware Middleware for Ubiquitous Greenhouses Using Wireless Sensor Networks. *Sensors (Basel, Switzerland)*, 11(5):4539–4561.

International Atomic Energy Agency (IAEA) (2008). A Practical Guide To Methods, Instrumentation and Sensor Technology. In *Field Estimation of Soil Water Content*, Training course series 30, page 141. IAEA, Vienna, Austria.

International Labour Office (2011). *Safety and health in agriculture*. Number 88,6,1 in Report in International Labour Conference. International Labour Office, Geneva.

International Telecommunication Union (2005). ITU Internet Reports - The Internet of Things.

ISO/IEC (2016). Information technology – Reference Architecture for Service Oriented Architecture (SOA RA) – Part 1: Terminology and concepts for SOA. Technical report ISO/IEC 18384-1:2016, ISO/IEC JTC 1/SC 38 Cloud Computing and Distributed Platforms, Switzerland.

Jackson, P. (1999). *Introduction to expert systems*. International computer science series. Addison-Wesley, Harlow, England, 3rd edition.

Janowicz, K., Haller, A., Cox, S. J., Le Phuoc, D., and Lefrançois, M. (2019). SOSA: A lightweight ontology for sensors, observations, samples, and actuators. *Journal of Web Semantics*, 56:1–10.

Jayaraman, P., Yavari, A., Georgakopoulos, D., Morshed, A., and Zaslavsky, A. (2016). Internet of Things Platform for Smart Farming: Experiences and Lessons Learnt. *Sensors*, 16(11):1884.

Jayaraman, P. P., Palmer, D., Zaslavsky, A., and Georgakopoulos, D. (2015). Do-it-Yourself Digital Agriculture applications with semantically enhanced IoT platform. In *IEEE Tenth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, pages 1–6, Singapore. IEEE.

Kamilaris, A., Gao, F., Prenafeta-Boldu, F. X., and Ali, M. I. (2016). Agri-IoT: A semantic framework for Internet of Things-enabled smart farming applications.

In *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, pages 442–447, Reston, Virginia, USA. IEEE.

Kubicek, H., Cimander, R., and Scholl, H. J. (2011). Layers of Interoperability. In *Organizational Interoperability in E-Government*, pages 85–96. Springer, Berlin, Heidelberg.

Lebo, T., Sahoo, S., McGuinness, D., Belhajjame, K., Cheney, J., Corsar, D., Garijo, D., Soiland-Reyes, S., Zednik, S., and Zhao, J. (2013). Prov-o: The prov ontology. *W3C recommendation*, 30.

Lefrançois, M. and Zimmermann, A. (2016). Supporting arbitrary custom datatypes in RDF and SPARQL. In *The Semantic Web. Latest Advances and New Domains - 13th International Conference, ESWC 2016*, pages 371–386, Heraklion, Crete, Greece.

Lori J. Abendroth, Roger W. Elmore, Matthew J. Boyer, and Stephanie K. Marlay (2011). *Corn Growth and Development*. Iowa State University, USA.

Maslow, A. H. (1943). A theory of human motivation. *Psychological review*, 50(4):370–396.

Migliaccio, K. W., Olczyk, T., Li, Y., Muñoz-Carpena, R., and Dispenza, T. (2002). Using Tensiometers for Vegetable Irrigation Scheduling in Miami-Dade County. Technical report ABE326, University of Florida, United States.

Muller, P.-A. and Gaertner, N. (2004). *Modélisation objet avec UML*. Eyrolles, Paris.

Musen, M. A. (2015). The protégé project: a look back and a look forward. *AI Matters*, 1(4):4–12.

Nadareishvili, I., Mitra, R., McLarty, M., and Amundsen, M. (2016). *Microservice architecture: aligning principles, practices, and culture*. O'Reilly Media, Inc, Sebastopol, CA, 1st edition.

Netherlands Study Centre for Technology Trends (2016). *The future of technology in agriculture*, volume 81. Stichting Toekomstbeeld der Techniek, The Hague, Netherlands.

Newman, S. (2015). *Building microservices: designing fine-grained systems*. O'Reilly Media, Beijing Sebastopol, CA, 1st edition.

Nguyen, Q.-D., Roussey, C., Poveda Villalón, M., De Vaulx, C., Chanet, J.-P., and Noûs, C. (2020a). CASO et IRRIG deux ontologies pour le développement de systèmes contextuels : cas d'usage sur l'automatisation de l'irrigation. In *31es Journées francophones d'Ingénierie des Connaissances*, Actes des 31e Journées francophones d'Ingénierie des Connaissances (IC 2020), pages 133–144, Angers, France. Sébastien Ferré.

Nguyen, Q.-D., Roussey, C., Poveda-Villalón, M., Vaulx, C. d., and Chanet, J.-P. (2020b). Development Experience of a Context-Aware System for Smart Irrigation Using CASO and IRRIG Ontologies. *Applied Sciences*, 10(5):1803.

Popovic, T., Latinovic, N., Pesic, A., Zecevic, Z., Krstajic, B., and Djukanovic, S. (2017). Architecting an IoT-enabled platform for precision agriculture and ecological monitoring: A case study. *Computers and Electronics in Agriculture*, 140:255–265.

Poveda-Villalón, M. (2012). A reuse-based lightweight method for developing linked data ontologies and vocabularies. In *The Semantic Web: Research and Applications*, pages 833–837, Berlin, Heidelberg. Springer.

Poveda-Villalón, M., Gómez-Pérez, A., and Suárez-Figueroa, M. C. (2014). OOPS! (OntOlogy Pitfall Scanner!): An On-line Tool for Ontology Evaluation. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 10(2):7–34.

Poveda-Villalón, M., Nguyen, Q.-D., Roussey, C., De Vaulx, C., and Chanet, J.-P. (2018a). Besoins ontologiques d'un système d'irrigation intelligent: comparaison entre SSN et SAREF. In *29es Journées Francophones d'Ingénierie des Connaissances, IC 2018*, pages 21–36, Nancy, France.

Poveda-Villalón, M., Nguyen, Q.-D., Roussey, C., de Vaulx, C., and Chanet, J.-P. (2018b). Ontological Requirement Specification for Smart Irrigation Systems: A SOSA/SSN and SAREF Comparison. In *Proceedings of the 9th International Semantic Sensor Networks Workshop, International Semantic Web Conference*, volume 2213 of *CEUR Workshop Proceedings*, pages 1–16, Monterey, United States.

Rahman, H. and Hussain, M. I. (2020). A comprehensive survey on semantic interoperability for Internet of Things: State-of-the-art and research challenges. *Transactions on Emerging Telecommunications Technologies*, pages 1–25.

Renting, H., Rossing, W., Groot, J., Van der Ploeg, J., Laurent, C., Perraud, D., Stobbelaar, D., and Van Ittersum, M. (2009). Exploring multifunctional

agriculture. A review of conceptual approaches and prospects for an integrative transitional framework. *Journal of Environmental Management*, 90:112–123.

Rijgersberg, H., Van Assem, M., and Top, J. (2013). Ontology of units of measure and related concepts. *Semantic Web*, 4(1):3–13.

Roussey, C., Chanet, J.-P., Soulignac, V., and Bernard, S. (2011a). Les ontologies en agriculture. *Ingénierie des systèmes d'information*, 16(3):55–84.

Roussey, C., Pinet, F., Kang, M. A., and Corcho, O. (2011b). An Introduction to Ontologies and Ontology Engineering. In *Ontologies in Urban Development Projects*, volume 1, pages 9–38. Springer, London.

Sha, K. and Shi, W. (2008). Consistency-driven Data Quality Management of Networked Sensor Systems. *J. Parallel Distrib. Comput.*, 68(9):1207–1221.

Soldatos, J., Kefalakis, N., Hauswirth, M., Serrano, M., Calbimonte, J.-P., Riahi, M., Aberer, K., Jayaraman, P. P., Zaslavsky, A., Žarko, I. P., Skorin-Kapov, L., and Herzog, R. (2015). OpenIoT: Open Source Internet-of-Things in the Cloud. In *Interoperability and Open-Source Solutions for the Internet of Things*, volume 9001, pages 13–25. Springer International Publishing, Cham.

Studer, R., Benjamins, V. R., and Fensel, D. (1998). Knowledge engineering: principles and methods. *Data & knowledge engineering*, 25(1-2):161–197.

Suárez-Figueroa, M. C., Gómez-Pérez, A., and Fernandez-Lopez, M. (2015). The neon methodology framework: A scenario-based methodology for ontology development. *Applied ontology*, 10(2):107–145.

Sun, J., De Sousa, G., Roussey, C., Chanet, J.-P., Pinet, F., and Hou, K.-M. (2016). Intelligent Flood Adaptive Context-aware System: How Wireless Sensors Adapt their Configuration based on Environmental Phenomenon Events. *Sensors & Transducers*, 206(11):68–81.

Sundmaeker, H., Guillemin, P., Friess, P., and Woelffle, S. (2010). *Vision and challenges for realising the Internet of Things*. Publications Office of the European Union, Luxembourg.

Sundmaeker, H., Verdouw, C., Wolfert, S., and Pérez Freire, L. (2016). Internet of food and farm 2020. *Digitising the Industry-Internet of Things connecting physical, digital and virtual worlds*, 49:129–151.

Taylor, K., Griffith, C., Lefort, L., Gaire, R., Compton, M., Wark, T., Lamb, D., Falzon, G., and Trotter, M. (2013). Farming the Web of Things. *IEEE Intelligent Systems*, 28(6):12–19.

Taylor, K. and Leidinger, L. (2011). Ontology-driven complex event processing in heterogeneous sensor networks. *The Semanic Web: Research and Applications*, 6644:285–299.

Urruty, N., Tailliez-Lefebvre, D., and Huyghe, C. (2016). Stability, robustness, vulnerability and resilience of agricultural systems. A review. *Agronomy for Sustainable Development*, 36(1):1–15.

Uschold, M. and Jasper, R. (1999). A framework for understanding and classifying ontology applications. In *Proceedings of the IJCAI-99 Workshop on Ontologies and Problem-Solving Methods (KRR5)*, pages 1–12, Stockholm, Sweden.

Van der Veer, H. and Wiles, A. (2008). Achieving technical interoperability.

Vasisht, D., Kapetanovic, Z., Won, J., Jin, X., Chandra, R., Sinha, S. N., Kapoor, A., Sudarshan, M., and Stratman, S. (2017). FarmBeats: An IoT Platform for Data-Driven Agriculture. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 515–529, Boston, Massachusetts, USA. USENIX Association.

W3C OWL Working Group (2012). OWL 2 Web Ontology Language Document Overview. W3C Recommendation, W3C.

Wolfert, S., Goense, D., and Sorensen, C. A. G. (2014). A Future Internet Collaboration Platform for Safe and Healthy Food from Farm to Fork. In *2014 Annual SRII Global Conference*, pages 266–273, San Jose, CA, USA. IEEE.

Yu, W., Liang, F., He, X., Hatcher, W. G., Lu, C., Lin, J., and Yang, X. (2018). A Survey on the Edge Computing for the Internet of Things. *IEEE Access*, 6:6900–6919.

# Appendix A

# New Vocabularies Defined in CASO and IRRIG

*"Five fingers, some are long, and some are short. But they are all united in one hand. "* – **Ho Chi Minh**

---

This appendix is a complement to Chapter 2 and Chapter 3. It presents the new vocabularies defined and presented in CASO and IRRIG v122019. The ontologists of TSCF will maintain and update these two ontologies. Their newest version will be published respectively in https://w3id.org/def/caso and https://w3id.org/def/irrig.

### A.1. New classes defined in CASO:

- **caso:Actuation** is a subclass of **sosa:Actuation**. It represents the act of carrying out an Actuation to change the state of world using an Actuator. This act is performed during the context exploitation phase of a context aware system. An Actuation links to an Actuator to describe what made the Actuation and how; links to an ActuatableProperty to describe the target that the Actuation change; links to a FeatureOfInterest to detail what that property was associated with; links to a Deduction by the wasInfluencedBy property when the act is performed based on the given Deduction; and links to the Function that is triggered by the Actuation by the triggersFunction property.
- **caso:Boundary** is the class that represents the limit of a State. The value of the limit could be a quantitative or a qualitative data. To store the value, the boundaryValue data property is used.
- **caso:Deduction** is a subclass of **caso:Observation**. It represents the act of carrying out an (Observation) Procedure to estimate the State of a Property

of a FeatureOfInterest. The Observation may be performed by an inference engine. A Deduction links to a State by the hasResultState property to describe what the result is; links to a TemporalEntity by the hasValidTime property to indicate when the result is valid and for how many time.

- **caso:Observation** is a subclass of **sosa:Observation**. It represents the act of carrying out an (Observation) Procedure to estimate or calculate a value of a Property of a FeatureOfInterest. This act can be performed during the context acquisition or context analysis phases of a context aware system. An Observation links to a Sensor to describe what made the Observation and how; links to an ObservableProperty to describe what the result is an estimate of; and links to a FeatureOfInterest to detail what that property was associated with. When the act is made during the context analysis phase the entry values of the procedure should be linked to the observation by the wasDerivedFrom property.
- **caso:Property** is a quality of an entity. An aspect of an entity that is intrinsic to and cannot exist without the entity.The possible quality values should be expressed by State. A Property is linked to its possible States using the hasState property.
- **caso:State** is a qualitative value of a Property, summarizing a set of information about that Property. A State links to its Property by the isStateOf property. To delimit the State of a Property, some Boundary may be defined.

### A.2. New object properties defined in CASO:

- `caso:greaterThan` is an object property. It represents the relation between states (**caso:State**) to define an order. In detail, the domain state is greater than the range state.
- `caso:greaterThanOrEqual` is an object property. It represents the relation between states (**caso:State**) to define an order. The domain state is greater than or equal to the range state.
- `caso:hasClosedLowerBoundary` is an object property. It represents the relation from a state (**caso:State**) to its lower boundary (**caso:Boundary**). The state is reached when the associated state value is equal or superior to the lower boundary value.
- `caso:hasClosedUpperBoundary` is an object property. It represents the relation from a state (**caso:State**) to its upper boundary (**caso:Boundary**). The state is reached when the associated state value is equal or inferior to the upper boundary value.
- `caso:hasOpenLowerBoundary` is an object property. It represents the relation

from a state (**caso:State**) to its lower boundary (**caso:Boundary**). The state is reached when the associated state value is superior to the lower boundary value.

- `caso:hasOpenUpperBoundary` is an object property. It represents the relation from a state (**caso:State**) to its upper boundary (**caso:Boundary**). The state is reached when the associated state value is inferior to the upper boundary value.
- `caso:hasResultState` is an object property. It represents the relation linking a deduction (**caso:Deduction**) and its result that is to say the state (**caso:State**) that is applied on a property for a time.
- `caso:hasState` is an object property. It represents the relation from a property to one of the possible states (**caso:State**) of that property.
- `caso:hasValidTime` is an object property. It represents the time that the result of a deduction applies to the property. Not necessarily the same as the `sosa:PhenomenonTime` or the sosa:ResultTime. May be an interval or an instant, or some other compound temporal entity.
- `caso:lesserThan` is an object property. It represents the relation between states (**caso:State**) to define an order. The domain state is lesser than the range state.
- `caso:lesserThanOrEqualTo` is an object property. It represents the relation between states (**caso:State**) to define an order. The domain state is lesser than or equal to the range state.
- `caso:triggersFunction` is an object property. It represents the relation from an actuation (**caso:Actuation**) to a function. The actuation trigger the operation of the function.

### A.3. New data properties defined in CASO:

- caso:boundaryValue is a data property. It represents the numeric value of a boundary (**caso:Boundary**).

### A.4. New classes defined in IRRIG:

- **irrig:AmountProperty** is a subclass of **ssn:Property**. It represents the amount or quantity aspect of an entity. The amount aspect of an entity is intrinsic and cannot exist without the entity. The value of the amount should be expressed by a qualitative value such as a state or by a quantitative value.
- **irrig:CropGrowthDeduction** is a subclass of **caso:Deduction**. It is a defined class that represents the act of carrying out a procedure to estimate the state of the growth property for the crop entity. The growth property is represented by the individual irrig:observableProperty_crop_growth. The crop entity is represented by the individual irrig:featureOfInterest_crop.

Two necessary and sufficient conditions are defined: (1) All instances of this class are linked to the individual irrig:observableProperty_crop_growth via the `sosa:observedProperty` object property; (2) All instances of this class are linked to the individual irrig:featureOfInterest_crop via the `sosa:hasFeatureOfInterest` object property. Additionally, each instance of this class is linked to an instance of the class **irrig:CropGrowthState** by the `caso:hasResultState` object property; is linked to an instance of the class **irrig:CropGrowthObservation** by the `prov:wasInformedBy` object property; and is linked to a sensor to describe the agent that made the deduction.

- **irrig:CropGrowthObservation** is a subclass of **caso:Observation**. It is a defined class that represents the act of carrying out a procedure (observation) to estimate the value of the growth property for the crop entity. The growth property is represented by the individual irrig:observableProperty_crop_growth. The crop entity is represented by the individual irrig:featureOfInterest_crop. Two necessary and sufficient conditions are defined: (1) All instances of this class are linked to the individual irrig:observableProperty_crop_growth via the `sosa:observedProperty` object property; (2) All instances of this class are linked to the individual irrig:featureOfInterest_crop via the `sosa:hasFeatureOfInterest` object property. Additionally, each instance of this class is linked to a sensor to describe the agent that made the observation.
- **irrig:CropGrowthState** is a subclass of **caso:State**. It represents the qualitative value of the crop growth property, which changes over time, summarizing a set of information about that property. To evaluate the state of the property, some boundaries need to be defined.
- **irrig:CropWaterNeedDeduction** is a subclass of **caso:Deduction**. It is a defined class that represents the act of carrying out a procedure to estimate the state of the water need property for the crop entity. The water need property is represented by the individual irrig:observableProperty_crop_waterNeed. The crop entity is represented by the individual irrig:featureOfInterest_crop. Two necessary and sufficient conditions are defined: (1) All instances of this class are linked to the individual irrig:observableProperty_crop_waterNeed via the `sosa:observedProperty` object property; (2) All instances of this class are linked to the individual irrig:featureOfInterest_crop via the `sosa:hasFeatureOfInterest` object property. Additionally, each instance of this class is linked to an instance of the class **irrig:CropWaterNeedState** by the `caso:hasResultState` object property; is linked to three instances of the classes **irrig:CropGrowthDeduction**, **irrig:RootZoneMoistureLevelDeduction** and **RainIntensityDeduction** by the `prov:wasInformedBy` object property; and

is linked to a sensor to describe the agent that made the deduction.

- **irrig:CropWaterNeedState** is a subclass of **caso:State**. It represents the qualitative value of the crop water need property, which changes over time, summarizing a set of information about that property. To evaluate the state of the property, some boundaries need to be defined.
- **irrig:DelayDurationObservation** is a subclass of **caso:Observation**. It is a defined class that represents the act of carrying out a procedure (observation) to calculate the number of delay days of the duration property for the delay entity. The duration property is represented by the individual irrig:observableProperty_delayDuration. The delay entity is represented by the individual irrig:featureOfInterest_delay. Two necessary and sufficient conditions are defined: (1) All instances of this class are linked to the individual irrig:observableProperty_delayDuration via the `sosa:observedProperty` object property; (2) All instances of this class are linked to the individual irrig:featureOfInterest_delay via the `sosa:hasFeatureOfInterest` object property. Additionally, each instance of this class is linked to a sensor to describe the agent that made the observation.
- **irrig:GrowthProperty** is a subclass of **ssn:Property**. It represents the growth aspect of an entity, that is, the physical development or the increasing of amount, number, or size of the entity. The growth aspect of an entity is intrinsic and cannot exist without the entity. The growth quality values should be expressed by a qualitative value such as a state or by a quantitative value.
- **irrig:IntensityProperty** is a subclass of **ssn:Property**. It represents the intensity aspect of an entity, that is, the strong effect or the level of power per unit area of the entity. The intensity aspect of an entity is intrinsic and cannot exist without the entity. The intensity quality values should be expressed by a qualitative value such as a state or by a quantitative value.
- **irrig:IrrigationActuation** is a subclass of **caso:Actuation**. It is a defined class that represents the act of carrying out an irrigation procedure (actuation) to change the state of a moisture actionable property of a soil entity using a watering actuator. The moisture actionable property is represented by the individual irrig:observableProperty_soil30cmDepth_moisture and/or irrig:observableProperty_soil60cmDepth_moisture. The soil entity is represented by the individual irrig:featureOfInterest_soil30cmDepth and/or irrig:featureOfInterest_soil60cmDepth. Two necessary and sufficient conditions are defined: (1) All instances of this class are linked to the individual irrig:observableProperty_soil30cmDepth_moisture and/or irrig:observableProperty_soil60cmDepth_moisture via the

`sosa:actsOnProperty` object property; (2) All instances of this class are linked to the individual irrig:featureOfInterest_soil30cmDepth and/or irrig:featureOfInterest_soil60cmDepth via the `sosa:hasFeatureOfInterest` object property. Additionally, each instance of this class is linked to a actuator to describe the agent that made the actuation.

- **irrig:MoistureProperty** is a subclass of **ssn:Property**. It represents the moisture aspect of an entity, that is, the amount of water available inside the entity. The moisture aspect of an entity is intrinsic and cannot exist without the entity. The moisture quality values should be expressed by a qualitative value such as a state or by a quantitative value.
- **irrig:RainDailyTotalQuantityObservation** is a subclass of **caso:Observation**. It is a defined class that represents the act of carrying out a procedure (observation) to estimate the value of the daily total quantity property for the rain entity. The daily total quantity property is represented by the individual irrig:observableProperty_rain_dailyTotalQuantity. The rain entity is represented by the individual irrig:featureOfInterest_rain. Two necessary and sufficient conditions are defined: (1) All instances of this class are linked to the individual irrig:observableProperty_rain_dailyTotalQuantity via the `sosa:observedProperty` object property; (2) All instances of this class are linked to the individual irrig:featureOfInterest_rain via the `sosa:hasFeatureOfInterest` object property. Additionally, each instance of this class is linked to a sensor to describe the agent that made the observation.
- **irrig:RainIntensityBoundary** represents the limit of a state of the rain intensity property. The value of the limit is quantitative. To store this value, the `irrig:boundaryValue` data property is used.
- **irrig:RainIntensityDeduction** is a subclass of **caso:Deduction**. It is a defined class that represents the act of carrying out a procedure to estimate the state of the intensity property for the rain entity. The intensity property is represented by the individual irrig:observableProperty_rain_intensity. The rain entity is represented by the individual irrig:featureOfInterest_rain. Two necessary and sufficient conditions are defined: (1) All instances of this class are linked to the individual irrig:observableProperty_rain_intensity via the `sosa:observedProperty` object property; (2) All instances of this class are linked to the individual irrig:featureOfInterest_rain via the `sosa:hasFeatureOfInterest` object property. Additionally, each instance of this class is linked to an instance of the class **irrig:RainIntensityState** by the `caso:hasResultState` object property; is linked to an instance of the class **irrig:RainDailyTotalQuantityObservation** by the `prov:wasInformedBy` object

property; and is linked to a sensor to describe the agent that made the deduction.

- **irrig:RainIntensityState** is a subclass of **caso:State**. It represents the qualitative value of the rain intensity property, which changes over time, summarizing a set of information about that property. To evaluate the state of the property, some boundaries need to be defined.
- **irrig:RainQuantityObservation** is a subclass of **caso:Observation**. It is a defined class that represents the act of carrying out a procedure (observation) to estimate the value of the quantity property for the rain entity. The quantity property is represented by the individual irrig:observableProperty_rain_quantity. The rain entity is represented by the individual irrig:featureOfInterest_rain. Two necessary and sufficient conditions are defined: (1) All instances of this class are linked to the individual irrig:observableProperty_rain_quantity via the `sosa:observedProperty` object property; (2) All instances of this class are linked to the individual irrig:featureOfInterest_rain via the `sosa:hasFeatureOfInterest` object property. Additionally, each instance of this class is linked to a sensor to describe the agent that made the observation.
- **irrig:RootZoneDailyMoistureObservation** is a subclass of **caso:Observation**. It is a defined class that represents the act of carrying out a procedure (observation) to estimate the value of the daily average moisture property for the root zone entity. The daily moisture property is represented by the individual irrig:observableProperty_rootZone_dailyAverageMoisture. The root zone entity is represented by the individual irrig:featureOfInterest_rootZone. Two necessary and sufficient conditions are defined: (1) All instances of this class are linked to the individual irrig:observableProperty_rootZone_dailyAverageMoisture via the `sosa:observedProperty` object property; (2) All instances of this class are linked to the individual irrig:featureOfInterest_rootZone via the `sosa:hasFeatureOfInterest` object property. Additionally, each instance of this class is linked to a sensor to describe the agent that made the observation.
- **irrig:RootZoneMoistureLevelBoundary** represents the limit of a state of the root zone moisture level property. The value of the limit is quantitative. To store this value, the `irrig:boundaryValue` data property is used.
- **irrig:RootZoneMoistureLevelDeduction** is a subclass of **caso:Deduction**. It is a defined class that represents the act of carrying out a procedure to estimate the state of the moisture level property for the root zone entity. The moisture level property is represented by the individual irrig:observableProperty_rootZone_moistureLevel. The root zone entity is represented by the individual irrig:featureOfInterest_rootZone. Two

necessary and sufficient conditions are defined: (1) All instances of this class are linked to the individual irrig:observableProperty_rootZone_moistureLevel via the `sosa:observedProperty` object property; (2) All instances of this class are linked to the individual irrig:featureOfInterest_rootZone via the `sosa:hasFeatureOfInterest` object property. Additionally, each instance of this class is linked to an instance of the class **irrig:RootZoneMoistureLevelState** by the `caso:hasResultState` object property; is linked to an instance of the class **irrig:RootZoneDailyAverageMoistureObservation** by the `prov:wasInformedBy` object property; and is linked to a sensor to describe the agent that made the deduction.

- **irrig:RootZoneMoistureLevelState** is a subclass of **caso:State**. It represents the qualitative value of the root zone moisture level property, which changes over time, summarizing a set of information about that property. To evaluate the state of the property, some boundaries need to be defined.
- **irrig:SleepingDurationObservation** is a subclass of **caso:Observation**. It is a defined class that represents the act of carrying out a procedure (observation) to calculate the number of sleeping days of the duration property for the sleeping entity. Sleeping days are the number of days that the system needs to wait until the next turn of doing an action. The duration property is represented by the individual irrig:observableProperty_sleepingDuration. The sleeping entity is represented by the individual irrig:featureOfInterest_sleeping. Two necessary and sufficient conditions are defined: (1) All instances of this class are linked to the individual irrig:observableProperty_sleepingDuration via the `sosa:observedProperty` object property; (2) All instances of this class are linked to the individual irrig:featureOfInterest_sleeping via the `sosa:hasFeatureOfInterest` object property. Additionally, each instance of this class is linked to a sensor to describe the agent that made the observation.
- **irrig:Soil30cmDepthDailyAverageMoistureObservation** is a subclass of **caso:Observation**. It is a defined class that represents the act of carrying out a procedure (observation) to estimate the value of the daily average moisture property for the soil at 30 cm depth entity. The daily average moisture property is represented by the individual irrig:observableProperty_soil30cmDepth_dailyAverageMoisture. The soil at 30 cm depth entity is represented by the individual irrig:featureOfInterest_soil30cmDepth. Two necessary and sufficient conditions are defined: (1) All instances of this class are linked to the individual irrig:observableProperty_soil30cmDepth_dailyAverageMoisture via the `sosa:observedProperty` object property; (2) All instances of this class

are linked to the individual irrig:featureOfInterest_soil30cmDepth via the `sosa:hasFeatureOfInterest` object property. Additionally, each instance of this class is linked to a sensor to describe the agent that made the observation.

- **irrig:Soil30cmDepthMoistureObservation** is a subclass of **caso:Observation**. It is a defined class that represents the act of carrying out a procedure (observation) to estimate the value of the moisture property for the soil at 30 cm depth entity. The daily average moisture property is represented by the individual irrig:observableProperty_soil30cmDepth_moisture. The soil at 30 cm depth entity is represented by the individual irrig:featureOfInterest_soil30cmDepth. Two necessary and sufficient conditions are defined: (1) All instances of this class are linked to the individual irrig:observableProperty_soil30cmDepth_moisture via the `sosa:observedProperty` object property; (2) All instances of this class are linked to the individual irrig:featureOfInterest_soil30cmDepth via the `sosa:hasFeatureOfInterest` object property. Additionally, each instance of this class is linked to a sensor to describe the agent that made the observation.
- **irrig:Soil60cmDepthDailyAverageMoistureObservation** is a subclass of **caso:Observation**. It is a defined class that represents the act of carrying out a procedure (observation) to estimate the value of the daily average moisture property for the soil at 60 cm depth entity. The daily average moisture property is represented by the individual irrig:observableProperty_soil60cmDepth_dailyAverageMoisture. The soil at 60 cm depth entity is represented by the individual irrig:featureOfInterest_soil60cmDepth. Two necessary and sufficient conditions are defined: (1) All instances of this class are linked to the individual irrig:observableProperty_soil60cmDepth_dailyAverageMoisture via the `sosa:observedProperty` object property; (2) All instances of this class are linked to the individual irrig:featureOfInterest_soil60cmDepth via the `sosa:hasFeatureOfInterest` object property. Additionally, each instance of this class is linked to a sensor to describe the agent that made the observation.
- **irrig:Soil60cmDepthMoistureObservation** is a subclass of **caso:Observation**. It is a defined class that represents the act of carrying out a procedure (observation) to estimate the value of the moisture property for the soil at 60 cm depth entity. The daily average moisture property is represented by the individual irrig:observableProperty_soil60cmDepth_moisture. The soil at 60 cm depth entity is represented by the individual irrig:featureOfInterest_soil60cmDepth. Two necessary and sufficient conditions are defined: (1) All instances of this class are linked to the individual irrig:observableProperty_soil60cmDepth_moisture via the `sosa:observedProperty` object property; (2) All instances of this

class are linked to the individual irrig:featureOfInterest_soil60cmDepth via the `sosa:hasFeatureOfInterest` object property. Additionally, each instance of this class is linked to a sensor to describe the agent that made the observation.

- **irrig:StressProperty** is a subclass of **ssn:Property**. It represents the stress aspect of an entity, that is, the pressure inside the entity. The stress aspect of an entity is intrinsic and cannot exist without the entity. The stress quality should be expressed by a qualitative value such as a state or by a quantitative value.
- **irrig:TemperatureProperty** is a subclass of **ssn:Property**. It represents the temperature aspect of an entity. The temperature aspect of an entity is intrinsic and cannot exist without the entity. The temperature quality should be expressed by a qualitative value such as a state or by a quantitative value.

### A.5. New individuals defined in IRRIG:

- irrig:actuator_watering_valve_1 represents an actuator n°1 that controls the valve of water.
- irrig:boundary_rain_quantity_high is a boundary line of division between two states heavy and moderate of the rain intensity property.
- irrig:boundary_rain_quantity_low is a boundary line of division between two states moderate and light of the rain intensity property.
- irrig:boundary_rain_quantity_maximum is a boundary line representing the maximal value that rain quantity cannot surpass.
- irrig:boundary_rain_quantity_minimum is a boundary line representing the minimal value that rain quantity cannot below.
- irrig:boundary_rootZone_tension_average is a boundary line of division between two states low and average of the root zone moisture level property.
- irrig:boundary_rootZone_tension_high is a boundary line of division between two states average and high of the root zone moisture level property.
- irrig:boundary_rootZone_tension_low is a boundary line of division between two states very low and low of the root zone moisture level property.
- irrig:boundary_rootZone_tension_maximum is a boundary line representing the maximal value that root zone daily average moisture cannot surpass.
- irrig:boundary_rootZone_tension_minimum is a boundary line representing the minimal value that root zone daily average moisture cannot below.
- irrig:boundary_rootZone_tension_saturation is a boundary line of division between two states very high and saturated of the root zone moisture level property.

- irrig:boundary_rootZone_tension_veryHigh is a boundary line of division between two states high and very high of the root zone moisture level property.
- irrig:boundary_rootZone_tension_veryLow is a boundary line of division between two states dry and very low of the root zone moisture level property.
- irrig:featureOfInterest_air is an instance of the class **sosa:FeatureOfInterest** that represents the air phenomenon.
- irrig:featureOfInterest_crop is an instance of the class **sosa:FeatureOfInterest** that represents the crop phenomenon.
- irrig:featureOfInterest_delay is an instance of the class **sosa:FeatureOfInterest** that represents the delay phenomenon.
- irrig:featureOfInterest_rain is an instance of the class **sosa:FeatureOfInterest** that represents the rain phenomenon.
- irrig:featureOfInterest_rootZone is an instance of the class **sosa:FeatureOfInterest** that represents the phenomenon of the root zone soil of a crop.
- irrig:featureOfInterest_sleeping is an instance of the class **sosa:FeatureOfInterest** that represents the phenomenon of waiting for the next action.
- irrig:featureOfInterest_soil30cmDepth is an instance of the class **sosa:FeatureOfInterest** that represents the soil at 30 cm depth phenomenon.
- irrig:featureOfInterest_soil60cmDepth is an instance of the class **sosa:FeatureOfInterest** that represents the soil at 60 cm depth phenomenon.
- irrig:observableProperty_air_dailyMaxTemperature is an instance of the classes **sosa:ObservableProperty** and **sosa:TemperatureProperty** that represents the maximal temperature property of the feature of interest air.
- irrig:observableProperty_air_dailyMinTemperature is an instance of the classes **sosa:ObservableProperty** and **sosa:TemperatureProperty** that represents the minimal temperature property of the feature of interest air.
- irrig:observableProperty_air_temperature is an instance of the classes **sosa:ObservableProperty** and **sosa:TemperatureProperty** that represents the temperature property of the feature of interest air.
- irrig:observableProperty_crop_growth is an instance of the classes **irrig:GrowthProperty** and **caso:Property** that represents the growth property of the feature of interest crop.
- irrig:observableProperty_crop_stage is an instance of the classes **irrig:GrowthProperty** and **caso:Property** that represents the stage property of

the feature of interest crop.

- irrig:observableProperty_crop_waterNeed is an instance of the classes **irrig:StressProperty** and **caso:Property** that represents the water need property of the feature of interest crop.
- irrig:observableProperty_delayDuration is an instance of the class **caso:Property** that represents the duration property of the feature of interest delay.
- irrig:observableProperty_rain_dailyTotalQuantity is an instance of the classes **irrig:AmountProperty** and **sosa:ObservableProperty** that represents the daily total quantity property of the feature of interest rain.
- irrig:observableProperty_rain_intensity is an instance of the classes **irrig:IntensityProperty** and **caso:Property** that represents the intensity property of the feature of interest rain.
- irrig:observableProperty_rain_quantity is an instance of the classes **irrig:IntensityProperty** and **caso:Property** that represents the intensity property of the feature of interest rain.
- irrig:observableProperty_rootZone_dailyAverageMoisture is an instance of the classes **irrig:MoistureProperty** and **sosa:ObservableProperty** that represents the daily average moisture property of the feature of interest root zone.
- irrig:observableProperty_rootZone_moistureLevel is an instance of the classes **irrig:MoistureProperty** and **caso:Property** that represents the moisture level property of the feature of interest root zone.
- irrig:observableProperty_sleepingDuration is an instance of the class **caso:Property** that represents the duration property of the feature of interest sleeping.
- irrig:observableProperty_soil30cmDepth_dailyAverageMoisture is an instance of the classes **irrig:MoistureProperty** and **sosa:ObservableProperty** that represents the daily average moisture property of the feature of interest soil at 30 cm depth.
- irrig:observableProperty_soil30cmDepth_moisture is an instance of the classes **irrig:MoistureProperty** and **sosa:ObservableProperty** that represents the moisture property of the feature of interest soil at 30 cm depth.
- irrig:observableProperty_soil60cmDepth_dailyAverageMoisture is an instance of the classes **irrig:MoistureProperty** and **sosa:ObservableProperty** that represents the daily average moisture property of the feature of interest soil at 60 cm depth.
- irrig:observableProperty_soil60cmDepth_moisture is an instance of the classes

**irrig:MoistureProperty** and **sosa:ObservableProperty** that represents the moisture property of the feature of interest soil at 60 cm depth.

- irrig:pluviometer_1 represents a pluviometer (actuator) n°1 that measures the rain quantity.
- irrig:pluviometer_2 represents a pluviometer (actuator) n°2 that measures the rain quantity.
- irrig:soil_tensiometer_1_30cm represents a tensiometer (actuator) n°1 that measures the soil moisture at 30 cm depth.
- irrig:soil_tensiometer_2_30cm represents a tensiometer (actuator) n°2 that measures the soil moisture at 30 cm depth.
- irrig:soil_tensiometer_3_30cm represents a tensiometer (actuator) n°3 that measures the soil moisture at 30 cm depth.
- irrig:soil_tensiometer_4_60cm represents a tensiometer (actuator) n°4 that measures the soil moisture at 60 cm depth.
- irrig:soil_tensiometer_5_60cm represents a tensiometer (actuator) n°5 that measures the soil moisture at 60 cm depth.
- irrig:soil_tensiometer_6_60cm represents a tensiometer (actuator) n°6 that measures the soil moisture at 60 cm depth.
- irrig:state_crop_growth_init is an instance of the class **caso:State** that represents the temporary undefined state of the crop growth property.
- irrig:state_crop_growth_r1 is an instance of the class **caso:State** that represents the crop after reaching "silking" stage of the crop.
- irrig:state_crop_growth_r1d15 is an instance of the class **caso:State** that represents the crop 20 days after reaching the "silking" stage.
- irrig:state_crop_growth_r5 is an instance of the class **caso:State** that represents the crop after reaching the "dent with 50% moisture" stage.
- irrig:state_crop_growth_r5hg45 is an instance of the class **caso:State** that represents the crop after reaching the "dent with 45% moisture" stage.
- irrig:state_crop_growth_v2 is an instance of the class **caso:State** that represents the crop after reaching the "7 leaves" stage.
- irrig:state_crop_growth_v7 is an instance of the class **caso:State** that represents the crop after reaching the "10 leaves" stage.
- irrig:state_crop_growth_v7d20 is an instance of the class **caso:State** that represents the crop 20 days after reaching the "10 leaves" stage.
- irrig:state_crop_waterNeed_init is an instance of the class **caso:State** that

represents the temporary undefined state of the crop water need property.

- irrig:state_crop_waterNeed_no is an instance of the class **caso:State** that represents the crop when it does not need water.
- irrig:state_crop_waterNeed_notApplicable is an instance of the class **caso:State** that represents the state corresponding to the crop water need property has not been evaluated yet.
- irrig:state_crop_waterNeed_unavailable is an instance of the class **caso:State** that represents the crop when it may need water soon.
- irrig:state_crop_waterNeed_yes is an instance of the class **caso:State** that represents the crop when it needs water.
- irrig:state_rain_intensity_heavy is an instance of the class **caso:State** that represents the rain intensity is heavy and can interrupt the working schedule.
- irrig:state_rain_intensity_init is an instance of the class **caso:State** that represents the temporary undefined state of the rain intensity property.
- irrig:state_rain_intensity_light is an instance of the class **caso:State** that represents the rain intensity is light and has no impact on the working schedule.
- irrig:state_rain_intensity_moderate is an instance of the class **caso:State** that represents the rain intensity is moderate and can change the working schedule.
- irrig:state_rootZone_moistureLevel_average is an instance of the class **caso:State** that represents the water in the root zone is just enough for the crop growth at state R1.
- irrig:state_rootZone_moistureLevel_dry is an instance of the class **caso:State** that represents the water in the root zone is not enough for the crop.
- irrig:state_rootZone_moistureLevel_high is an instance of the class **caso:State** that represents the water in the root zone is just enough for the crop growth at state V7d20
- irrig:state_rootZone_moistureLevel_init is an instance of the class **caso:State** that represents the temporary undefined state of the root zone moisture level property.
- irrig:state_rootZone_moistureLevel_low is an instance of the class **caso:State** that represents the water in the root zone is just enough for the crop growth at state R1d15.
- irrig:state_rootZone_moistureLevel_saturated is an instance of the class **caso:State** that represents the water in the root zone is saturated for the crop.
- irrig:state_rootZone_moistureLevel_veryHigh is an instance of the class

**caso:State** that represents the water in the root zone is just enough for the crop growth at state V7.

- irrig:state_rootZone_moistureLevel_veryLow is an instance of the class **caso:State** that represents the water in the root zone is just enough for the crop growth at state R5.

## A.6. New rules defined in IRRIG:

**Table 32**: Rule for the property CropGrowth to reach the V7d20 state

| **Code:** | **Full name:** |
|---|---|
| ADv122019-CG-V7d20 | ArvalisData-IrrigVersion122019-CropGrowth-V7d20 |

**Description:**
The goal of this rule is to determine the state of CropGrowth. The rule implements the transition from the state V7 or V7d20 to the state V7d20. The input of this rule is the farmer observation at V7 (?observation_crop_growth). The output of this rule is the Crop Growth deduction at V7d20 (?deduction_crop_growth). The mechanism of this rule checks if the duration (?duration_observation_deduction) between the farmer observation and the deduction is superior or equal to a time threshold (?value_Th_Time_20D), then the result of the deduction is V7d20.

**Rule in SWRL:**

```
irrig:CropGrowthObservation(?observation_crop_growth) ^
sosa:hasResult(?observation_crop_growth, ?irrig:state_crop_growth_v7) ^
time:Instant(?instant_observation) ^
sosa:phenomenonTime(?observation_crop_growth, ?instant_observation) ^
time:inXSDDateTimeStamp(?instant_observation, ?timestamp_observation) ^

irrig:CropGrowthDeduction(?deduction_crop_growth) ^
prov:wasInformedBy(?deduction_crop_growth, ?observation_crop_growth) ^
time:Interval(?interval_deduction) ^
sosa:phenomenonTime(?deduction_crop_growth, ?interval_deduction) ^
time:hasBeginning(?interval_deduction, ?instant_begin_deduction) ^
time:inXSDDateTimeStamp(?instant_begin_deduction, ?timestamp_begin_deduction) ^

temporal:duration(?duration_observation_deduction, ?timestamp_begin_deduction,
                                                   ?timestamp_observation,"Minutes") ^

caso:hasOpenUpperBoundary(irrig:state_crop_growth_v7, ?Th_Time_20D) ^
caso:boundaryValue(?Th_Time_20D, ?value_Th_Time_20D) ^

swrlb:greaterThanOrEqual(?duration_observation_deduction, ?value_Th_Time_20D)

-> caso:hasResultState(?deduction_crop_growth, irrig:state_crop_growth_v7d20)
```

**Table 33**: Rule for the property CropGrowth to reach the R1 state

| **Code:** | **Full name:** |
|---|---|
| ADv122019-CG-R1 | ArvalisData-IrrigVersion122019-CropGrowth-R1 |

**Description:**
The goal of this rule is to determine the state of CropGrowth. The rule implements the transition from the state V7d20 or R1 to the state R1. The input of this rule is the farmer observation at R1 (?observation_crop_growth). The output of this rule is the Crop Growth deduction at R1 (?deduction_crop_growth). The mechanism of this rule checks if the duration (?duration_observation_deduction) between the farmer observation and the deduction is inferior to a time threshold (?value_Th_Time_15D), then the result of the deduction is R1.

**Rule in SWRL:**

```
irrig:CropGrowthObservation(?observation_crop_growth) ^
sosa:hasResult(?observation_crop_growth, ?irrig:state_crop_growth_r1) ^
time:Instant(?instant_observation) ^
sosa:phenomenonTime(?observation_crop_growth, ?instant_observation) ^
time:inXSDDateTimeStamp(?instant_observation, ?timestamp_observation) ^

irrig:CropGrowthDeduction(?deduction_crop_growth) ^
prov:wasInformedBy(?deduction_crop_growth, ?observation_crop_growth) ^
time:Interval(?interval_deduction) ^
sosa:phenomenonTime(?deduction_crop_growth, ?interval_deduction) ^
time:hasBeginning(?interval_deduction, ?instant_begin_deduction) ^
time:inXSDDateTimeStamp(?instant_begin_deduction, ?timestamp_begin_deduction) ^

temporal:duration(?duration_observation_deduction, ?timestamp_begin_deduction,
                                                   ?timestamp_observation, "Minutes") ^
caso:hasOpenUpperBoundary(irrig:state_crop_growth_r1, ?Th_Time_15D) ^
caso:boundaryValue(?Th_Time_15D, ?value_Th_Time_15D) ^
swrlb:lessThan(?duration_observation_deduction, ?value_Th_Time_15D)

-> caso:hasResultState(?deduction_crop_growth, irrig:state_crop_growth_r1)
```

**Table 34**: Rule for the property CropGrowth to reach the R1d15 state

| **Code:** | **Full name:** |
|---|---|
| ADv122019-CG-R1d15 | ArvalisData-IrrigVersion122019-CropGrowth-R1d15 |

**Description:**
The goal of this rule is to determine the state of CropGrowth. The rule implements the transition from the state R1 or R1d15 to the state R1d15. The input of this rule is the farmer observation at R1d15 (?observation_crop_growth). The output of this rule is the Crop Growth deduction at R1d15 (?deduction_crop_growth). The mechanism of this rule checks if the duration (?duration_observation_deduction) between the farmer observation and the deduction is superior or equal to a time threshold (?value_Th_Time_15D), then the result of the deduction is R1d15.

**Rule in SWRL:**
irrig:CropGrowthObservation(?observation_crop_growth) ˆ
sosa:hasResult(?observation_crop_growth, ?irrig:state_crop_growth_r1) ˆ
time:Instant(?instant_observation) ˆ
sosa:phenomenonTime(?observation_crop_growth, ?instant_observation) ˆ
time:inXSDDateTimeStamp(?instant_observation, ?timestamp_observation) ˆ

irrig:CropGrowthDeduction(?deduction_crop_growth) ˆ
prov:wasInformedBy(?deduction_crop_growth, ?observation_crop_growth) ˆ
time:Interval(?interval_deduction) ˆ
sosa:phenomenonTime(?deduction_crop_growth, ?interval_deduction) ˆ
time:hasBeginning(?interval_deduction, ?instant_begin_deduction) ˆ
time:inXSDDateTimeStamp(?instant_begin_deduction, ?timestamp_begin_deduction) ˆ

temporal:duration(?duration_observation_deduction, ?timestamp_begin_deduction,
?timestamp_observation,"Minutes") ˆ

caso:hasOpenUpperBoundary(irrig:state_crop_growth_r1, ?Th_Time_15D) ˆ
caso:boundaryValue(?Th_Time_15D, ?value_Th_Time_15D) ˆ

swrlb:greaterThanOrEqual(?duration_observation_deduction, ?value_Th_Time_15D)

-> caso:hasResultState(?deduction_crop_growth, irrig:state_crop_growth_r1d15)

**Table 35**: Rule for the property CropGrowth to reach the R5hg45 state

| **Code:** ADv122019-CG-R5hg45 | **Full name:** ArvalisData-IrrigVersion122019-CropGrowth-R5hg45 |
|---|---|
| **Description:** The goal of this rule is to determine the state of CropGrowth. The rule implements the transition from the state R5 or R5hg45 state to the state R5hg45. The input of this rule is the farmer observation at R5hg45 (?observation_crop_growth). The output of this rule is the Crop Growth deduction at R5hg45 (?deduction_crop_growth). This rule's mechanism is to check if the last farmer observation is at R5hg45 state, then the deduction is at R5hg45. | |
| **Rule in SWRL:**<br>irrig:CropGrowthObservation(?observation_crop_growth) ˆ<br>sosa:hasResult(?observation_crop_growth, ?irrig:state_crop_growth_r5hg45) ˆ<br><br>irrig:CropGrowthDeduction(?deduction_crop_growth) ˆ<br>prov:wasInformedBy(?deduction_crop_growth, ?observation_crop_growth) ˆ<br><br>-> caso:hasResultState(?deduction_crop_growth, irrig:state_crop_growth_r5hg45) | |

**Table 36**: Rule for the property CropWaterNeed to reach the No state (part 2)

| **Code:** | **Full name:** |
|---|---|
| ADv122019-CWN-C2-P2 | ArvalisData-IrrigVersion122019-CropWaterNeed-Condition-2-Part-2 |

**Description:**
The goal of this rule is to determine the state of CropWaterNeed. The rule implements the transition from the state Init state to the state NonApplicable. The input of this rule is: (1) the value of the SleepingDuration property of d-1 (?sleeping_duration_yesterday); (2) the state of the CropGrowth property for the d day (?state_crop_growth); (3) the state of RootZoneMoistureLevel for the d day. The output of this rule is: the CropWaterNeed property is at the state No for the d day. The mechanism of this rule is: to check if the value of SleepingDuration is 0, the state of CropGrowth is V7d20 and the state of RootZoneMoistureLevel is superior to Average.

**Rule in SWRL:**

```
irrig:CropWaterNeedDeduction(?deduction_crop_waterneed) ^

irrig:SleepingDurationObservation(?observation_sleeping_duration_yesterday) ^
prov:wasInformedBy(?deduction_crop_waterneed,
                                        ?observation_sleeping_duration_yesterday) ^
sosa:hasResult(?observation_sleeping_duration_yesterday, arvalis:result_value_0_days) ^

irrig:CropGrowthDeduction(?deduction_crop_growth) ^
prov:wasInformedBy(?deduction_crop_waterneed, ?deduction_crop_growth) ^
caso:hasResultState(?deduction_crop_growth, irrig:state_crop_growth_v7d20) ^

irrig:RootZoneMoistureLevelDeduction(?deduction_rootzone_moisturelevel) ^
prov:wasInformedBy(?deduction_crop_waterneed, ?deduction_rootzone_moisturelevel) ^
caso:hasResultState(?deduction_rootzone_moisturelevel, ?state_rootzone_moisturelevel) ^
caso:greaterThan(?state_rootzone_moisturelevel, irrig:state_rootZone_moistureLevel_average)

-> caso:hasResultState(?deduction_crop_waterneed, irrig:state_crop_waterNeed_no)
```

**Table 37**: Rule for the property CropWaterNeed to reach the No state (part 3)

| **Code:** | **Full name:** |
|---|---|
| ADv122019-CWN-C2-P3 | ArvalisData-IrrigVersion122019-CropWaterNeed-Condition-2-Part-3 |

**Description:**
The goal of this rule is to determine the state of CropWaterNeed. The rule implements the transition from the state Init to the state NonApplicable. The input of this rule is: (1) the value of the SleepingDuration property of d-1 (?sleeping_duration_yesterday); (2) the state of the CropGrowth property for the d day (?state_crop_growth); (3) the state of RootZoneMoistureLevel for the d day. The output of this rule is: the CropWaterNeed property is at the state No for the d day. The mechanism of this rule is: to check if the value of SleepingDuration is 0, the state of CropGrowth is R1 and the state of RootZoneMoistureLevel is superior to Low.

**Rule in SWRL:**
irrig:CropWaterNeedDeduction(?deduction_crop_waterneed) ^
irrig:SleepingDurationObservation(?observation_sleeping_duration_yesterday) ^
prov:wasInformedBy(?deduction_crop_waterneed,
?observation_sleeping_duration_yesterday) ^
sosa:hasResult(?observation_sleeping_duration_yesterday, arvalis:result_value_0_days) ^

irrig:CropGrowthDeduction(?deduction_crop_growth) ^
prov:wasInformedBy(?deduction_crop_waterneed, ?deduction_crop_growth) ^
caso:hasResultState(?deduction_crop_growth, irrig:state_crop_growth_r1) ^
irrig:RootZoneMoistureLevelDeduction(?deduction_rootzone_moisturelevel) ^
prov:wasInformedBy(?deduction_crop_waterneed, ?deduction_rootzone_moisturelevel) ^
caso:hasResultState(?deduction_rootzone_moisturelevel, ?state_rootzone_moisturelevel) ^
caso:greaterThan(?state_rootzone_moisturelevel, irrig:state_rootZone_moistureLevel_low)

-> caso:hasResultState(?deduction_crop_waterneed, irrig:state_crop_waterNeed_no)

**Table 38**: Rule for the property CropWaterNeed to reach the No state (part 4)

| **Code:** | **Full name:** |
|---|---|
| ADv122019-CWN-C2-P4 | ArvalisData-IrrigVersion122019-CropWaterNeed-Condition-2-Part-4 |

**Description:**
The goal of this rule is to determine the state of CropWaterNeed. The rule implements the transition from the Init state to the NonApplicable. The input of this rule is: (1) the value of the SleepingDuration property of d-1 (?sleeping_duration_yesterday); (2) the state of the CropGrowth property for the d day (?state_crop_growth); (3) the state of RootZoneMoistureLevel for the d day. The output of this rule is: the CropWaterNeed property is at the state No for the d day. The mechanism of this rule is: to check if the value of SleepingDuration is 0, the state of CropGrowth is R1d15 and the state of RootZoneMoistureLevel is superior to VeryLow.

**Rule in SWRL:**
irrig:CropWaterNeedDeduction(?deduction_crop_waterneed) ^
irrig:SleepingDurationObservation(?observation_sleeping_duration_yesterday) ^
prov:wasInformedBy(?deduction_crop_waterneed,
?observation_sleeping_duration_yesterday) ^
sosa:hasResult(?observation_sleeping_duration_yesterday, arvalis:result_value_0_days) ^

irrig:CropGrowthDeduction(?deduction_crop_growth) ^
prov:wasInformedBy(?deduction_crop_waterneed, ?deduction_crop_growth) ^
caso:hasResultState(?deduction_crop_growth, irrig:state_crop_growth_r1d15) ^
irrig:RootZoneMoistureLevelDeduction(?deduction_rootzone_moisturelevel) ^
prov:wasInformedBy(?deduction_crop_waterneed, ?deduction_rootzone_moisturelevel) ^
caso:hasResultState(?deduction_rootzone_moisturelevel, ?state_rootzone_moisturelevel) ^
caso:greaterThan(?state_rootzone_moisturelevel, irrig:state_rootZone_moistureLevel_veryLow)

-> caso:hasResultState(?deduction_crop_waterneed, irrig:state_crop_waterNeed_no)

**Table 39**: Rule for the property CropWaterNeed to reach the No state (part 5)

| **Code:** | **Full name:** |
|---|---|
| ADv122019-CWN-C2-P5 | ArvalisData-IrrigVersion122019-CropWaterNeed-Condition-2-Part-5 |

**Description:**
The goal of this rule is to determine the state of CropWaterNeed. The rule implements the transition from the Init state to the NonApplicable. The input of this rule is: (1) the value of the SleepingDuration property of d-1 (?sleeping_duration_yesterday); (2) the state of the CropGrowth property for the d day (?state_crop_growth); (3) the state of RootZoneMoistureLevel for the d day. The output of this rule is: the CropWaterNeed property is at the state No for the d day. The mechanism of this rule is: to check if the value of SleepingDuration is 0, the state of CropGrowth is R5 and the state of RootZoneMoistureLevel is superior to Dry.

**Rule in SWRL:**

```
irrig:CropWaterNeedDeduction(?deduction_crop_waterneed) ^

irrig:SleepingDurationObservation(?observation_sleeping_duration_yesterday) ^
prov:wasInformedBy(?deduction_crop_waterneed,
                                        ?observation_sleeping_duration_yesterday) ^
sosa:hasResult(?observation_sleeping_duration_yesterday, arvalis:result_value_0_days) ^

irrig:CropGrowthDeduction(?deduction_crop_growth) ^
prov:wasInformedBy(?deduction_crop_waterneed, ?deduction_crop_growth) ^
caso:hasResultState(?deduction_crop_growth, irrig:state_crop_growth_r5) ^

irrig:RootZoneMoistureLevelDeduction(?deduction_rootzone_moisturelevel) ^
prov:wasInformedBy(?deduction_crop_waterneed, ?deduction_rootzone_moisturelevel) ^
caso:hasResultState(?deduction_rootzone_moisturelevel, ?state_rootzone_moisturelevel) ^
caso:greaterThan(?state_rootzone_moisturelevel, irrig:state_rootZone_moistureLevel_dry)

-> caso:hasResultState(?deduction_crop_waterneed, irrig:state_crop_waterNeed_no)
```

**Table 40**: Rule for the property CropWaterNeed to reach the Yes state (part 2)

| **Code:** | **Full name:** |
|---|---|
| ADv122019-CWN-C3-P2 | ArvalisData-IrrigVersion122019-CropWaterNeed-Condition-3-Part-2 |

**Description:**
The goal of this rule is to determine the state of CropWaterNeed. The rule implements the transition from the Init state to the Yes. The input of this rule is: (1) the value of the SleepingDuration property of d-1 (?sleeping_duration_yesterday); (2) the state of the CropGrowth property for the d day (?state_crop_growth); (3) the state of RootZoneMoistureLevel for the d day. The output of this rule is: the CropWaterNeed property is at the state Yes for the d day. The mechanism of this rule is: to check if the value of SleepingDuration is 0, the state of CropGrowth is V7d20 and the state of RootZoneMoistureLevel is inferior or equal to Average.

**Rule in SWRL:**
irrig:CropWaterNeedDeduction(?deduction_crop_waterneed) ^
irrig:SleepingDurationObservation(?observation_sleeping_duration_yesterday) ^
prov:wasInformedBy(?deduction_crop_waterneed,
?observation_sleeping_duration_yesterday) ^
sosa:hasResult(?observation_sleeping_duration_yesterday, arvalis:result_value_0_days) ^
irrig:CropGrowthDeduction(?deduction_crop_growth) ^
prov:wasInformedBy(?deduction_crop_waterneed, ?deduction_crop_growth) ^
caso:hasResultState(?deduction_crop_growth, irrig:state_crop_growth_v7d20) ^
irrig:RootZoneMoistureLevelDeduction(?deduction_rootzone_moisturelevel) ^
prov:wasInformedBy(?deduction_crop_waterneed, ?deduction_rootzone_moisturelevel) ^
caso:hasResultState(?deduction_rootzone_moisturelevel, ?state_rootzone_moisturelevel) ^
caso:lesserThanOrEqualTo(?state_rootzone_moisturelevel,
irrig:state_rootZone_moistureLevel_average)

-> caso:hasResultState(?deduction_crop_waterneed, irrig:state_crop_waterNeed_yes)

**Table 41**: Rule for the property CropWaterNeed to reach the Yes state (part 3)

| **Code:** | **Full name:** |
|---|---|
| ADv122019-CWN-C3-P3 | ArvalisData-IrrigVersion122019-CropWaterNeed-Condition-3-Part-3 |

**Description:**
The goal of this rule is to determine the state of CropWaterNeed. The rule implements the transition from the Init state to the Yes. The input of this rule is: (1) the value of the SleepingDuration property of d-1 (?sleeping_duration_yesterday); (2) the state of the CropGrowth property for the d day (?state_crop_growth); (3) the state of RootZoneMoistureLevel for the d day. The output of this rule is: the CropWaterNeed property is at the state Yes for the d day. The mechanism of this rule is: to check if the value of SleepingDuration is 0, the state of CropGrowth is R1 and the state of RootZoneMoistureLevel is inferior or equal to Low.

**Rule in SWRL:**
irrig:CropWaterNeedDeduction(?deduction_crop_waterneed) ^
irrig:SleepingDurationObservation(?observation_sleeping_duration_yesterday) ^
prov:wasInformedBy(?deduction_crop_waterneed,
?observation_sleeping_duration_yesterday) ^
sosa:hasResult(?observation_sleeping_duration_yesterday, arvalis:result_value_0_days) ^
irrig:CropGrowthDeduction(?deduction_crop_growth) ^
prov:wasInformedBy(?deduction_crop_waterneed, ?deduction_crop_growth) ^
caso:hasResultState(?deduction_crop_growth, irrig:state_crop_growth_r1) ^
irrig:RootZoneMoistureLevelDeduction(?deduction_rootzone_moisturelevel) ^
prov:wasInformedBy(?deduction_crop_waterneed, ?deduction_rootzone_moisturelevel) ^
caso:hasResultState(?deduction_rootzone_moisturelevel, ?state_rootzone_moisturelevel) ^
caso:lesserThanOrEqualTo(?state_rootzone_moisturelevel,
irrig:state_rootZone_moistureLevel_low)

-> caso:hasResultState(?deduction_crop_waterneed, irrig:state_crop_waterNeed_yes)

**Table 42**: Rule for the property CropWaterNeed to reach the Yes state (part 4)

| **Code:** | **Full name:** |
|---|---|
| ADv122019-CWN-C3-P4 | ArvalisData-IrrigVersion122019-CropWaterNeed-Condition-3-Part-4 |

**Description:**
The goal of this rule is to determine the state of CropWaterNeed. The rule implements the transition from the Init state to the Yes. The input of this rule is: (1) the value of the SleepingDuration property of d-1 (?sleeping_duration_yesterday); (2) the state of the CropGrowth property for the d day (?state_crop_growth); (3) the state of RootZoneMoistureLevel for the d day. The output of this rule is: the CropWaterNeed property is at the state Yes for the d day. The mechanism of this rule is: to check if the value of SleepingDuration is 0, the state of CropGrowth is R1d15 and the state of RootZoneMoistureLevel is inferior or equal to VeryLow.

**Rule in SWRL:**

```
irrig:CropWaterNeedDeduction(?deduction_crop_waterneed) ^

irrig:SleepingDurationObservation(?observation_sleeping_duration_yesterday) ^
prov:wasInformedBy(?deduction_crop_waterneed,
                                        ?observation_sleeping_duration_yesterday) ^
sosa:hasResult(?observation_sleeping_duration_yesterday, arvalis:result_value_0_days) ^

irrig:CropGrowthDeduction(?deduction_crop_growth) ^
prov:wasInformedBy(?deduction_crop_waterneed, ?deduction_crop_growth) ^
caso:hasResultState(?deduction_crop_growth, irrig:state_crop_growth_r1d15) ^

irrig:RootZoneMoistureLevelDeduction(?deduction_rootzone_moisturelevel) ^
prov:wasInformedBy(?deduction_crop_waterneed, ?deduction_rootzone_moisturelevel) ^
caso:hasResultState(?deduction_rootzone_moisturelevel, ?state_rootzone_moisturelevel) ^
caso:lesserThanOrEqualTo(?state_rootzone_moisturelevel,
                                        irrig:state_rootZone_moistureLevel_veryLow)

-> caso:hasResultState(?deduction_crop_waterneed, irrig:state_crop_waterNeed_yes)
```

**Table 43**: Rule for the property CropWaterNeed to reach the Yes state (part 5)

| **Code:** | **Full name:** |
|---|---|
| ADv122019-CWN-C3-P5 | ArvalisData-IrrigVersion122019-CropWaterNeed-Condition-3-Part-5 |

**Description:**
The goal of this rule is to determine the state of CropWaterNeed. The rule implements the transition from the Init state to the Yes. The input of this rule is: (1) the value of the SleepingDuration property of d-1 (?sleeping_duration_yesterday); (2) the state of the CropGrowth property for the d day (?state_crop_growth); (3) the state of RootZoneMoistureLevel for the d day. The output of this rule is: the CropWaterNeed property is at the state Yes for the d day. The mechanism of this rule is: to check if the value of SleepingDuration is 0, the state of CropGrowth is R5 and the state of RootZoneMoistureLevel is equal to Dry.

**Rule in SWRL:**
irrig:CropWaterNeedDeduction(?deduction_crop_waterneed) ^

irrig:SleepingDurationObservation(?observation_sleeping_duration_yesterday) ^
prov:wasInformedBy(?deduction_crop_waterneed,
?observation_sleeping_duration_yesterday) ^
sosa:hasResult(?observation_sleeping_duration_yesterday, arvalis:result_value_0_days) ^

irrig:CropGrowthDeduction(?deduction_crop_growth) ^
prov:wasInformedBy(?deduction_crop_waterneed, ?deduction_crop_growth) ^
caso:hasResultState(?deduction_crop_growth, irrig:state_crop_growth_r5) ^

irrig:RootZoneMoistureLevelDeduction(?deduction_rootzone_moisturelevel) ^
prov:wasInformedBy(?deduction_crop_waterneed, ?deduction_rootzone_moisturelevel) ^
caso:hasResultState(?deduction_rootzone_moisturelevel, ?state_rootzone_moisturelevel) ^
caso:lesserThanOrEqualTo(?state_rootzone_moisturelevel,
irrig:state_rootZone_moistureLevel_dry)

-> caso:hasResultState(?deduction_crop_waterneed, irrig:state_crop_waterNeed_yes)

**Table 44**: Rule for the property SleepingDuration to get a value (part 1)

| **Code:** | **Full name:** |
|---|---|
| ADv122019-SD-P1 | ArvalisData-IrrigVersion122019-SleepingDuration-Part-1 |
| **Description:**<br>The goal of this rule is to determine the state of SleepingDuration. The rule implements the calculation of the value of SleepingDuration value The input of this rule is: the state of the CropWaterNeed property of the current day (?deduction_crop_waterneed). The output of this rule is: the SleepingDuration property with a value equal to 0 days. The mechanism of this rule is: to check if the value of CropWaterNeed equals to the state notApplicable. | |
| **Rule in SWRL:**<br>irrig:SleepingDurationObservation(?observation_sleeping_duration) ^<br>irrig:CropWaterNeedDeduction(?deduction_crop_waterneed) ^<br>prov:wasInformedBy(?observation_sleeping_duration,?deduction_crop_waterneed) ^<br>caso:hasResultState(?deduction_crop_waterneed, irrig:state_crop_waterNeed_notApplicable)<br><br>-> sosa:hasResult(?observation_sleeping_duration, arvalis:result_value_0_days) | |

**Table 45**: Rule for the property SleepingDuration to get a value (part 2)

| **Code:** | **Full name:** |
|---|---|
| ADv122019-SD-P2 | ArvalisData-IrrigVersion122019-SleepingDuration-Part-2 |
| **Description:**<br>The goal of this rule is to determine the state of SleepingDuration. The rule implements the calculation of the value of SleepingDuration value The input of this rule is: the state of the CropWaterNeed property of the current day (?deduction_crop_waterneed). The output of this rule is: the SleepingDuration property with a value equal to 0 days. The mechanism of this rule is: to check if the value of CropWaterNeed equals to the state no. | |
| **Rule in SWRL:**<br>irrig:SleepingDurationObservation(?observation_sleeping_duration) ˆ<br>irrig:CropWaterNeedDeduction(?deduction_crop_waterneed) ˆ<br>prov:wasInformedBy(?observation_sleeping_duration, ?deduction_crop_waterneed) ˆ<br>caso:hasResultState(?deduction_crop_waterneed, irrig:state_crop_waterNeed_no)<br><br>-> sosa:hasResult(?observation_sleeping_duration, arvalis:result_value_0_days) | |

**Table 46**: Rule for the property SleepingDuration to get a value (part 3)

| **Code:** | **Full name:** |
|---|---|
| ADv122019-SD-P3 | ArvalisData-IrrigVersion122019-SleepingDuration-Part-3 |
| **Description:**<br>The goal of this rule is to determine the state of SleepingDuration. The rule implements the calculation of the value of SleepingDuration value The input of this rule is: the state of the CropWaterNeed property of the current day (?deduction_crop_waterneed). The output of this rule is: the SleepingDuration property with a value equal to 6 days. The mechanism of this rule is: to check if the value of CropWaterNeed equals to the state yes. | |
| **Rule in SWRL:**<br>irrig:SleepingDurationObservation(?observation_sleeping_duration) ˆ<br>irrig:CropWaterNeedDeduction(?deduction_crop_waterneed) ˆ<br>prov:wasInformedBy(?observation_sleeping_duration,?deduction_crop_waterneed) ˆ<br>caso:hasResultState(?deduction_crop_waterneed, irrig:state_crop_waterNeed_yes)<br><br>-> sosa:hasResult(?observation_sleeping_duration, arvalis:result_value_6_days) | |

**Table 47**: Rule for the property SleepingDuration to get a value (part 4)

| **Code:** | **Full name:** |
|---|---|
| ADv122019-SD-P4 | ArvalisData-IrrigVersion122019-SleepingDuration-Part-4 |

**Description:**

The goal of this rule is to determine the state of SleepingDuration. The rule implements the calculation of the value of SleepingDuration value The input of this rule is: (1) the state of the CropWaterNeed property of the current day (?deduction_crop_waterneed), (2) the value of SleepingDuration property of the last day, and (3) the value of the DelayDuration property of the current day (?delay_duration). The output of this rule is: the SleepingDuration property with a value equal to the minimum between two values: (1) the maximum sleep duration, and (2) the sum of the value of the SleepingDuration of the last day and the number delay duration minus one day. The mechanism of this rule is: to check if the value of CropWaterNeed equals to the unavailable state; to compare the value of the maximum of sleeping duration which equals to 15 days, and the calculation that the value of the SleepingDuration of the last day (?sleeping_duration_yesterday) plus the value of the delay duration of the current day (?delay_duration) minus one day, and get the minimum value between them (?value_result_get_sleeping)

**Rule in SWRL:**

irrig:SleepingDurationObservation(?observation_sleeping_duration) ^
irrig:CropWaterNeedDeduction(?deduction_crop_waterneed) ^
prov:wasInformedBy(?observation_sleeping_duration,?deduction_crop_waterneed) ^
caso:hasResultState(?deduction_crop_waterneed, irrig:state_crop_waterNeed_unavailable) ^

irrig:SleepingDurationObservation(?observation_sleeping_duration_yesterday) ^
prov:wasInformedBy(?observation_sleeping_duration,
?observation_sleeping_duration_yesterday) ^
sosa:hasResult(?observation_sleeping_duration_yesterday, ?sleeping_duration_yesterday) ^
om:hasNumericalValue(?sleeping_duration_yesterday, ?value_sleeping_duration_yesterday) ^

irrig:DelayDurationObservation(?observation_delay_duration) ^
prov:wasInformedBy(?observation_sleeping_duration, ?observation_delay_duration) ^
sosa:hasResult(?observation_delay_duration, ?delay_duration) ^
om:hasNumericalValue(?delay_duration, ?value_delay_duration) ^

sosa:Result(?result_day) ^
om:hasUnit(?result_day, time:unitDay) ^
om:hasNumericalValue(?result_day, ?value_result_day) ^
swrlm:eval(?time_tmp,
"value_sleeping_duration_yesterday + value_delay_duration - 1",
?value_sleeping_duration_yesterday,?value_delay_duration) ^
swrlm:eval(?value_result_get_slleeping, "if(time_tmp < 15, time_tmp, 15)",?time_tmp) ^
swrlb:equal(?value_result_day, ?value_result_get_sleeping)

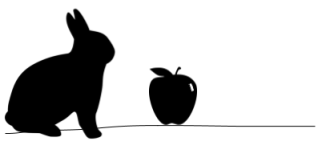
-> sosa:hasResult(?observation_sleeping_duration, ?result_day)

**Table 48**: Rule for automatically inferring GreaterThan relations

| **Code:** ADv122019-TransGreaterThan | **Full name:** ArvalisData-IrrigVersion122019-TransferGreaterThan |
|---|---|
| **Description:** This rule aims to infer automatically GreaterThan relations between states. This rule is a bonus besides the 24 major rules of IRRIG. | |
| **Rule in SWRL:**<br>caso:State(?x) ^<br>caso:State(?y) ^<br>caso:State(?z) ^<br>caso:greaterThan(?x,?y) ^<br>caso:greaterThan(?y,?z)<br><br>-> caso:greaterThan(?x,?z) | |

**Table 49**: Rule for automatically inferring LesserThan relations

| **Code:** ADv122019-TransLesserThan | **Full name:** ArvalisData-IrrigVersion122019-TransferLesserThan |
|---|---|
| **Description:** This rule aims to infer automatically LesserThan relations between states. This rule is a bonus besides the 24 major rules of IRRIG. | |
| **Rule in SWRL:**<br>caso:State(?x) ^<br>caso:State(?y) ^<br>caso:State(?z) ^<br>caso:LesserThan(?x,?y) ^<br>caso:LesserThan(?y,?z)<br><br>-> caso:LesserThan(?x,?z) | |



# Appendix B

# 14-Pages Summary in French

*Les sciences ont deux extrémités qui se touchent : la première est la pure ignorance naturelle où se trouvent tous les hommes en naissant ; l'autre extrémité est celle où arrivent les grandes âmes, qui, ayant parcouru tout ce que les hommes peuvent savoir, trouvent qu'ils ne savent rien, et se rencontrent en cette même ignorance d'où ils étaient partis.* – **Blaise Pascal**

---

Cette annexe a pour vocation de récapituler le contenu de cette thèse en français. Dans la limite de 14 pages, elle est organisée en cinq parties suivant les cinq sections principales de cette dissertation : une introduction, trois chapitres et une conclusion. L'introduction explique le contexte, les objectifs et introduit globalement les contributions de la thèse. Le premier chapitre porte sur la première contribution qui est une architecture basée sur le principe de microservice. Cette architecture permets aux développeurs d'un système de concentrer leurs efforts sur l'objectif du service plutôt que sur ses aspects logiciels et matériels. Le deuxième chapitre se concentre sur la deuxième contribution qui est une ontologie intitulée CASO. Cette ontologie est une collection de vocabulaire pour modéliser les données hétérogènes générées par le système contextuel. De plus, elle inclut un mécanisme pour créer des règles de raisonnement. Le troisième chapitre porte sur la troisième contribution: un système d'aide à la décision pour un système contextuel d'irrigation de l'équipe TSCF d'INRAE. Finalement, la conclusion donne une partie des perspectives ouvertes par la thèse.

### Introduction

Le 21e siècle fait face à un triple défi sur le plan agricole : la sécurité alimentaire est

menacée par la croissance démographique, le changement climatique et la rareté de ressources. Pour ces raisons, augmenter le rendement et la production agricole est une nécessité indéniable. Une solution des solutions est d'appliquer les technologies informatiques dans des activités agricoles afin d'en améliorer les performances. L'agriculture basée sur l'informatique est connue sous le nom e-agriculture. Dans la e-agriculture, les agriculteurs utilisent un système de management agricole pour réaliser des activités agricoles.

Les développements récents des technologies ont engendré un nouveau domaine de la e-agriculture qui s'appelle l'Agriculture 4.0. Le principe de l'Agriculture 4.0 est d'utiliser les technologies de l'Internet des Objets pour transformer un système de management agricole en un système contextuel agricole. L'Internet des Objets est un scénario dans lequel les personnes et les objets sont connectés indépendamment du temps, du lieu et du réseau informatique *(International Telecommunication Union, 2005)*. Les technologies de l'Internet des Objets incluent toutes les technologies qui contribuent à ce scénario. Un système contextuel agricole est un système avec la capacité de réagir convenablement au contexte. La figure 59 décrit un système contextuel agricole. Ce système est composé d'un système d'aide à la décision, d'une passerelle, d'un actionneur et d'un réseau de capteurs. Le réseau de capteurs collecte des données environnementales et les envoie au système d'aide à la décision via la passerelle. Ce système produit des décisions basées sur ces données. Il contrôle l'actionneur avec des commandes.

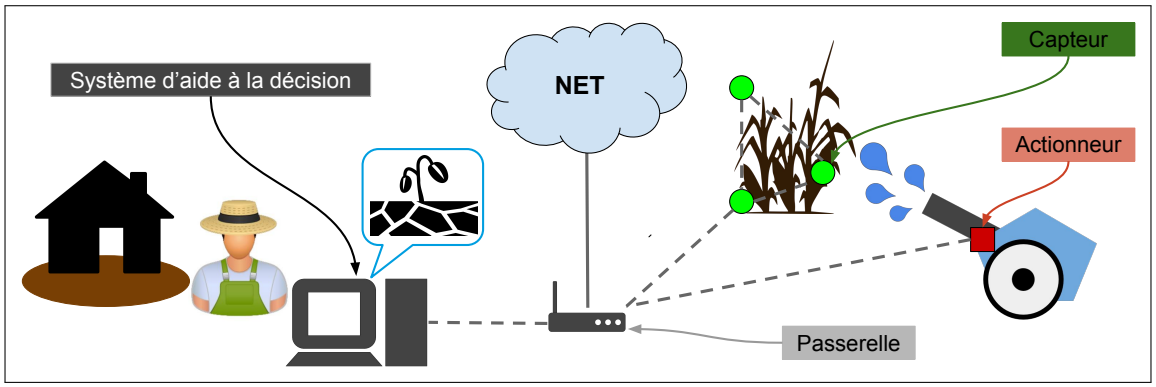



**Figure 59**: Un prototype de système contextuel pour l'irrigation

La conception d'un système contextuel agricole nécessite de relever deux défis spécifiques à l'agriculture. Premièrement, l'agriculture est une activité saisonnière, avec un lieu de travail externe, ce qui implique plusieurs facteurs imprévisibles qui influent sur les aspects logiciels et matériels du système. Le système peut donc être amené à être mis à jour régulièrement. Ce premier défi est lié à l'évolutivité du système. Deuxièmement, l'hétérogénéité de données générées par ce système est un

obstacle dans l'interopérabilité entre les différents appareils composant un même, ou l'interopérabilité entre plusieurs systèmes différents dans l'écosystème de l'Internet des Objets. Ce défi est lié à l'interopérabilité du système.

Cette thèse a trois objectifs. Le premier objectif est de surmonter le défi d'évolutivité. Cette thèse propose une architecture basée sur le principe de microservice qui permet aux développeurs d'un système de se concentrer sur ses services plutôt que sur les aspects logiciels et matériels. Le deuxième objectif est lié au défi d'interopérabilité. Dans ce cadre, la thèse propose une ontologie, intitulée CASO, qui fournit un vocabulaire pour modéliser les données générées par les systèmes contextuels. De plus, elle inclut un mécanisme de création de règles de raisonnement. Le troisième objectif révèle de la pratique. La contribution associée est la transformation d'une méthode d'irrigation manuelle intitulée IRRINOV® en un système d'irrigation automatique pour l'équipe TSCF d'INRAE. Le développement de ce système repose sur les deux premières contributions de cette thèse. Les expériences menées après le développement de ce système sont aussi une contribution scientifique.

## B.1. Architecture de Systèmes Contextuels Utilisant une Pile de Services

Un système contextuel est *« un système qui utilise le contexte pour fournir des informations et des services appropriés à l'utilisateur. Il convient de noter que la pertinence d'une information ou d'un service dépend de la tâche réalisée par l'utilisateur » (Abowd et al., 1999)*. Le contexte dans ce type de système est *« un ensemble des entités caractérisées par leur état, plus toute information permettant de dériver les changements d'états de ces entités » (Sun et al., 2016)*. Deux types de contexte sont définis :

- le contexte de bas niveau contient des données quantitatives telles que les mesures issues de capteurs ou les résultats d'agrégation sur ces mesures.
- le contexte de haut niveau, quant à lui, est constitué des données qualitatives synthétisant la situation d'une entité.

Le cycle de fonctionnement d'un système contextuel se découpe en quatre phases, qui sont : (1) l'acquisition du contexte, (2) la modélisation du contexte, (3) l'analyse du contexte et (4) l'exploitation du contexte. La figure 60 illustre le cycle de fonctionnement d'un système contextuel dédié à l'irrigation.

- phase d'acquisition du contexte : au cours de cette phase, le système acquiert des données provenant de diverses sources. La principale source de données est le réseau de capteurs sans fil qui mesure, collecte et transmet ses mesures sous forme de flux. Les sorties de cette phase sont les mesures, leurs métadonnées et toutes données nécessaires à la prise de décision.
- phase de modélisation du contexte : les mesures sont annotées pour pouvoir être

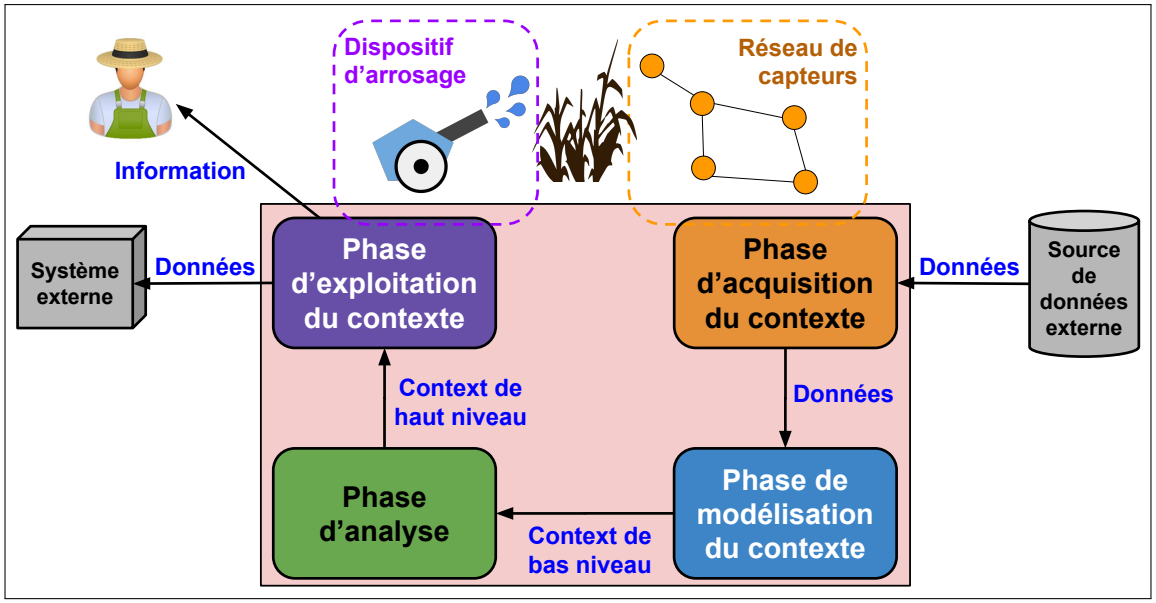



**Figure 60**: Le cycle de fonctionnement d'un système contextuel dédié à l'irrigation

intégrées dans un modèle de données commun. Les ontologies sont la solution choisie pour définir ce modèle. Ces données stockées et organisées constituent le contexte de bas niveau.

- phase d'analyse : au cours de cette phase, le contexte est enrichi. Tout d'abord des agrégations temporelles et spatiales sont appliquées sur les mesures des capteurs. Ainsi, le contexte de bas niveau s'enrichit de nouvelles données. Ensuite, un raisonnement est appliqué sur le contexte de bas niveau afin de produire le contexte de haut niveau.
- phase d'exploitation du contexte : les décisions ayant été prises à la phase précédente, il faut maintenant les mettre en application. Dans cette phase, le système exploite le contexte de haut niveau et le distribue à différents agents : des appareils connectés.

Il est possible de regrouper tous les services d'un système contextuel en quatre groupes correspondants aux quatre phases de son cycle de fonctionnement. Ce fait s'appuie sur le principe de microservices : *« un système est composé par des services, où chaque service est petit, fortement découplé, orienté tâche et prend en charge l'interopérabilité grâce à une communication basée sur les messages »* *(Nadareishvili et al., 2016; Newman, 2015)*. Cette approche possède trois avantages pour le dévelopement d'un système contextuel. Premièrement, la mise à niveau du système est indépendante du logiciel et du matériel, car celui-ci se concentre sur les services. Deuxièmement, les développeurs du système ont plusieurs choix, car cette approche

peut générer un grand nombre de solutions de conceptualisation, pour un même nombre fix de services. Troisièmement, le partage des connaissances sur un nombre défini de services dans la communauté est plus facile que le partage de connaissances sur u nombre indéfini de composants logiciels et matériels.

Suivant les remarques ci-dessus, cette thèse propose une pile de services pour les systèmes contextuels qui inclut 16 services. Ces 16 services sont extraits d'une étude sur plusieurs systèmes contextuels agricoles et des connaissances des experts participant à ce projet. Il est à noter que les systèmes contextuels dans les différents domaines partagent toujours un même groupe de processus. Les développeurs d'un système contextuel peuvent utiliser les services dans cette pile dont ils ont besoin pour construire leur système complet. La figure 61 illustre la pile de services pour les systèmes contextuels.

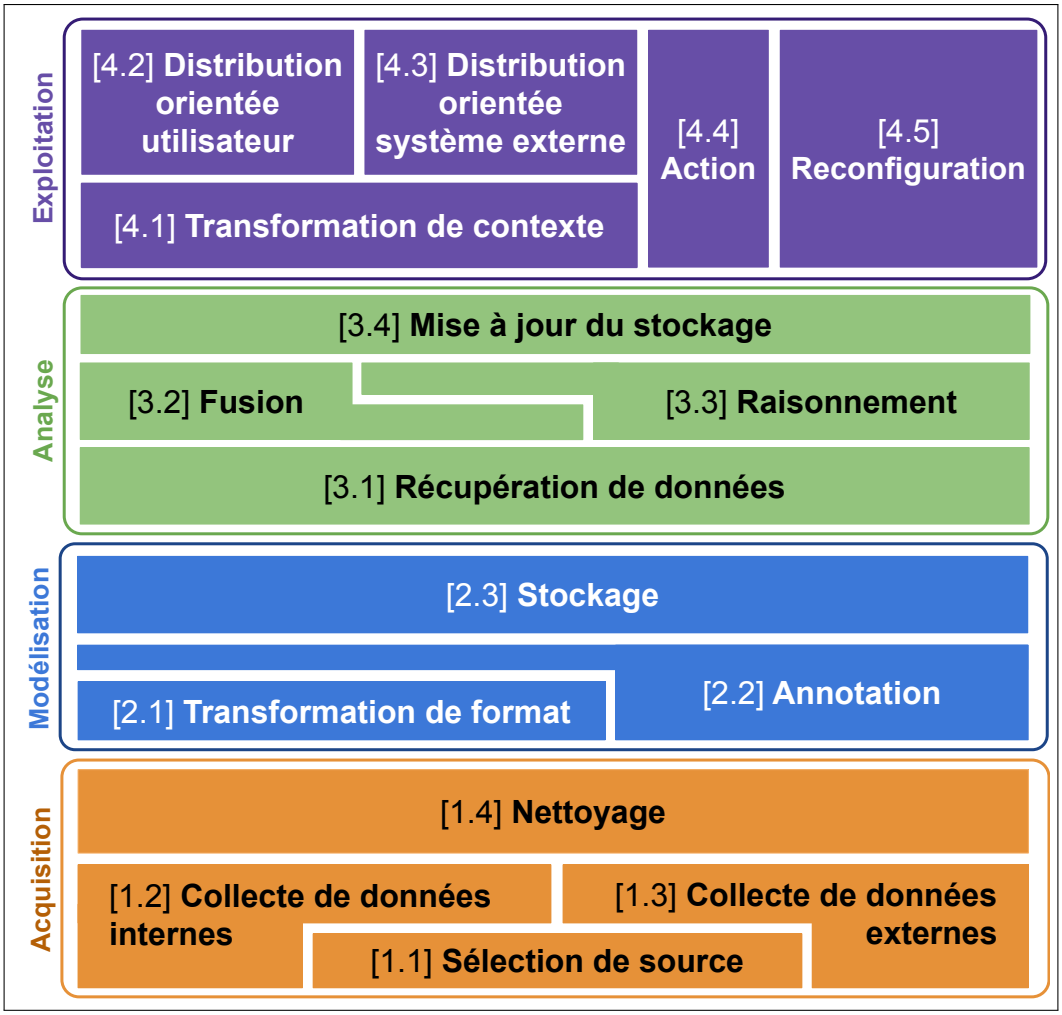



**Figure 61**: La pile de services pour les systèmes contextuels

- sélection de source : ce service sert à enregistrer les informations de configuration des sources au moment de l'exécution;
- collecte de données internes : ce service sert à collecter de données à partir de sources de données internes;
- collecte de données externes : ce service sert à collecter de données à partir de sources de données externes;
- nettoyage : ce service sert à garantir l'exactitude des données d'entrée;
- transformation de format : ce service sert à transformer le format original des données en un autre format;
- annotation : ce service sert à interpréter de données à l'aide d'un schéma spécifique ou de méta-données;
- stockage : ce service sert à stocker les données;
- récupération de données : ce service sert à récupérer les données de contexte de bas niveau stockées dans le système;
- fusion : ce service sert à utiliser des opérateurs d'agrégation ou des techniques de fusion pour combiner plusieurs types de données de contexte de bas niveau afin de produire de nouvelles données de contexte de bas niveau;
- raisonnement : ce service sert à utiliser des techniques de raisonnement pour produire de nouvelles données contextuelles de haut niveau;
- mise à jour du stockage : ce service sert à mettre à jour les nouvelles données produites après la phase d'analyse dans le stockage;
- transformation de contexte : ce service sert à transformer le format des données de contexte de haut niveau reçues de la phase d'analyse en un autre format;
- distribution orientée utilisateur : ce service sert à fournir des informations dans un format lisible par les utilisateurs;
- distribution orientée système externe : ce service sert à envoyer les messages encapsulés reçus du service de transformation de contexte aux systèmes externes;
- action : ce service sert à contrôler un dispositif d'actionnement;
- reconfiguration : ce service sert à reconfigurer automatiquement le planning de travail des autres services en fonction des changements de contexte.

Pour vérifier si la pile de services est capable de couvrir les systèmes contextuels, cette thèse propose un état de l'art de six systèmes agricoles intelligents : *le système Kirby Smart Farm*, *le système du projet PLANTS*, *le système Phenonet-OpenIoT*, *le système FarmBeats*, *le système du projet BIO-ICT* dédié à l'irrigation et *le système de Hwang et al.* Dans cet état de l'art, chaque système agricole intelligent est analysé

en utilisant la pile de services. La table 50 récapitule le résultat de cette analyse.

| | | Kirby Smart Farm | PLANTS | Phenonet-OpenIoT | FarmBeats | BIO-ICT | Hwang et al. |
|---|---|---|---|---|---|---|---|
| **Acquisition** | 1.1. Sélection de source | x | | x | | x | |
| | 1.2. Collecte de données internes | x | x | x | x | x | x |
| | 1.3. Collecte de données externes | x | x | x | | | |
| | 1.4. Nettoyage | x | x | x | x | x | x |
| **Modélisation** | 2.1. Transformation de format | x | | | | | |
| | 2.2. Annotation | x | x | x | x | x | x |
| | 2.3. Stockage | x | x | x | x | x | x |
| **Analyse** | 3.1. Récupération de données | x | x | x | x | x | x |
| | 3.2. Fusion | x | | x | x | x | x |
| | 3.3. Raisonnement | x | x | x | x | x | x |
| | 3.4. Mise à jour du stockage | x | x | x | x | x | x |
| **Exploitation** | 4.1. Transformation de contexte | x | x | x | x | x | x |
| | 4.2. Distribution orientée utilisateur | x | x | x | x | x | x |
| | 4.3. Distribution orientée système externe | x | | | x | x | |
| | 4.4. Action | | x | | x | | x |
| | 4.6. Reconfiguration | | | | | | |

**Table 50**: Services disponibles dans les six systèmes agricoles intelligents

Suivant l'état de l'art, il est possible de conclure que la pile de services peut couvrir tous les systèmes contextuels. De plus, on voit aussi que certains services sont inclus dans tous les systèmes et donc essentiels, alors que certains autres sont optionnels.

## B.2. Ontologie de Systèmes Contextuels pour l'Hétérogénéité de Données

Une ontologie est un modèle abstrait qui modélise un phénomène dans le monde *(Studer et al., 1998)*. Une ontologie doit être lisible par machine, et être acceptée et

partagée dans une communauté. De plus, une ontologie peut prendre diverses formes en fonction de leur objectif *(Uschold and Jasper, 1999)*. Dans le cadre de ce projet, deux types d'ontologie sont considérés : les ontologies de données et les ontologies logiques *(Roussey et al., 2011a)*. Les ontologies de données ont pour objectif de modéliser des données, tandis que les ontologies logiques proposent aussi des règles qui soutiennent le raisonnement.

Cette thèse utilise l'ontologie comme un outil pour l'interopérabilité sémantique entre des dispositifs du système ou entre ce système et des systèmes externes. Plus précisément, nous proposons une ontologie pour les systèmes contextuels, intitulée CASO, est proposée. Cette ontologie répond au besoin d'un vocabulaire qui couvre les exigences de base des systèmes contextuels en général. Ces exigences de base sont les différents types de calculs (mesure, agrégation et déduction) et leurs données (données mesurées, données agrégées et données inférées). La figure 62 illustre l'ontologie CASO. Le vocabulaire de CASO contient au total 27 classes, 40 propriétés d'objets et quatre propriétés de type de données.

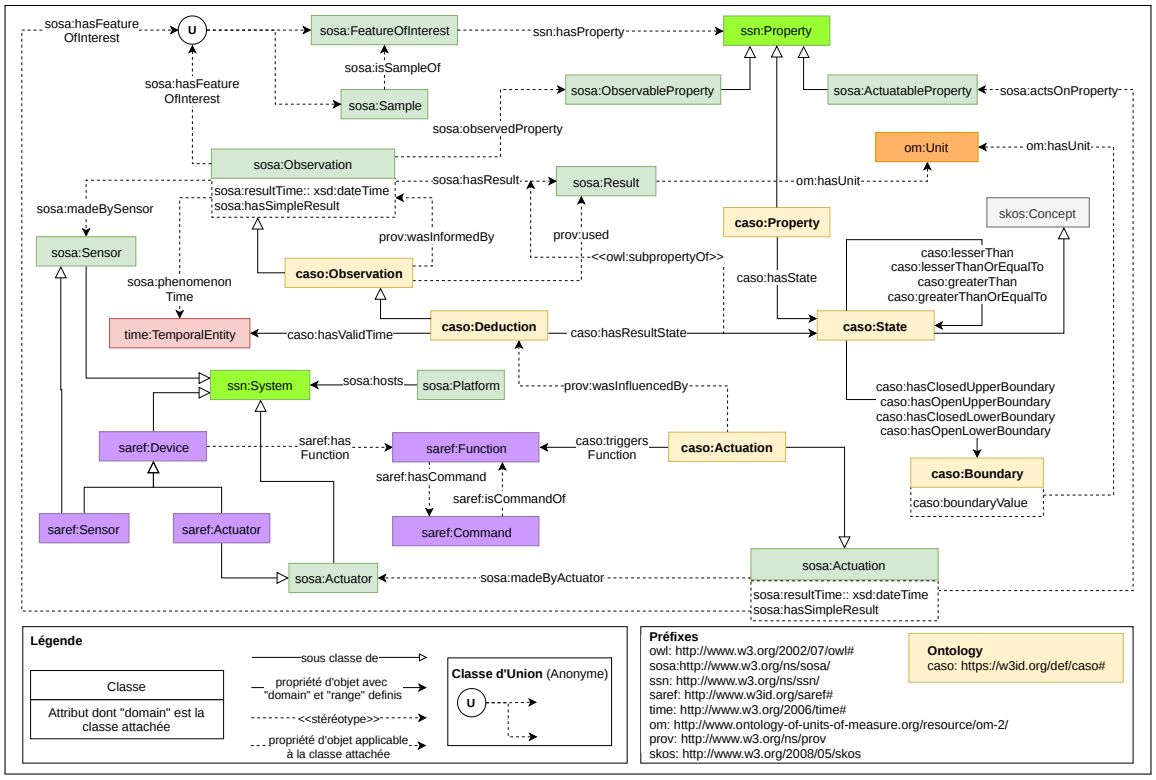



**Figure 62**: L'ontologie CASO

CASO réutilise les vocabulaires des plusieurs ontologies. Il est à noter que la plupart des ontologies sélectionnées sont développées et maintenues par des organisations de normalisation crédibles.

- la nouvelle version de l'ontologie Semantic Sensor Network (SOSA/SSN) qui est une recommandation du W3C et de l'OGC. Cette ontologie décrit les capteurs, leurs mesures, les échantillons et les actionneurs *(Janowicz et al., 2019)*.
- l'ontologie Smart Appliances REFerence (SAREF) est une recommandation de l'ETSI pour résoudre les problèmes d'interopérabilité dans le domaine de l'IoT *(ETSI TS 103 264 - v2.1.1, 2017)*. Cette ontologie modélise les appareils connectés ainsi que leurs fonctions, commandes, services, états et profils.
- l'ontologie PROV du W3C est utilisée pour modéliser la provenance des données *(Lebo et al., 2013)*.
- l'ontologie SKOS du W3C est préconisée pour représenter les thésaurus et tout système d'organisation des connaissances *(Bechhofer and Miles, 2009)*. Un thésaurus peut être utilisé pour identifier les types de mesures ou de phénomènes observés.
- l'ontologie Time du W3C permet de décrire tous les éléments temporels.
- l'ontologie Units of Measure and Related Concepts (OM) identifie un ensemble d'unités de mesures *(Rijgersberg et al., 2013)*.

De plus, CASO définit plusieurs classes et propriétés qui lui donnent trois avantages en comparaison avec les ontologies existantes. Premièrement, elle précise l'observation en définissant sa propre classe observation et une nouvelle classe déduction. En conséquence, il a trois niveaux d'observation différents pour décrire les mesures, les agrégations et les déductions. Deuxièmement, CASO définit une nouvelle propriété d'objet et une nouvelle classe pour le résultat d'une déduction. Alors que les résultats d'observation et d'agrégation sont des valeurs numériques, les résultats de déductions sont des états. Troisièmement, CASO possède des nouvelles propriétés et classes pour préciser les valeurs d'un état. Il y a toujours un lien entre un état et ses valeurs numériques correspondantes. Ce troisième avantage implique aussi un mécanisme pour définir des règles de raisonnement.

Afin d'évaluer l'ontologie CASO, nous utilisons OPPS!, un outil de détecter des pièges et des erreurs dans une ontologie *(Poveda-Villalón et al., 2014)*. Cet outil est disponible gratuitement en ligne : http://oops.linkeddata.es/. Ensuite, CASO est publiée sur le GitHub : https://w3id.org/def/caso. Ce fait non seulement permet à notre équipe de promouvoir CASO et la partager à la communauté, mais aussi est une méthode d'améliorer cette ontologie au fur et à mesure en recevant des commentaires des autres utilisateurs.

### B.3. Développement d'un Système d'Aide à la Décision pour l'Irrigation

Un système d'aide à la décision est une partie du système contextuel de TSCF.

Le fonctionnement de ce système s'appuie sur une méthode manuelle d'irrigation nommée IRRINOV®. Nous avons converti cette méthode en un système d'irrigation automatique.

La méthode IRRINOV® est développée par l'institut technique Arvalis et ses partenaires. Cette méthode propose un guide de bonnes pratiques aux agriculteurs afin de décider des dates d'irrigation en fonction des tensiomètres et des pluviomètres. Cette méthode fournit des conseils pour répondre à trois questions : (1) quand l'irrigation devrait-elle commencer ? C'est-à-dire quand l'agriculteur doit-il mettre en place son dispositif d'arrosage sur la parcelle (2) quand lancer un arrosage c'est-à-dire quand démarrer un tour d'eau 3 ? (3) quand l'irrigation devrait-elle s'arrêter ? Autrement dit, quand l'agriculteur peut-il retirer son dispositif d'arrosage de la parcelle ? l'agriculteur peut-il retirer son dispositif d'arrosage de la parcelle ? La méthode IRRINOV® est constituée d'un ensemble de tables de décision et de recommandations pour gérer l'irrigation d'une parcelle. Cette méthode propose de nombreuses variantes selon le type de sol de la parcelle et de sa culture. Dans le cadre de ce projet, la méthode IRRINOV® pour la culture du maïs grain en sol argilo-calcaire *(Arvalis et al., 2007)* a été retenue pour être mise en œuvre dans le système d'aide à l'irrigation. Les agriculteurs utilisent les tables de décision pour déterminer le démarrage d'un tour d'eau. Le choix de la table dépend du type de sol, de la durée du tour d'eau, de la culture et de son stade de développement. Cependant, chaque décision s'appuie de l'humidité du sol, la quantité d'eau de pluie et le stade de développement de la culture. Notez qu'il y a deux façons de déterminer le stade de développement de la culture : (1) par l'observation d'un expert (humain) ou (2) par la température de l'atmosphère. Dans le cadre de ce projet, la première façon est sélectionnée. Un objectif de cette thèse est de traduire ces tables de décision en règles de raisonnement et d'utiliser ces règles pour faire fonctionner un système d'aide à l'irrigation.

Afin de développer le système d'aide à l'irrigation, une nouvelle méthodologie de développement est proposée dans ce projet. Cette méthodologie est une combinaison des méthodologies mini-cascade et LOT. La méthodologie mini-cascade s'inspire de la méthodologie cascade, qui définit du développement d'un système en cinq activités: (1) spécification, (2) conception, (3) mise en œuvre, (4) validation et (5) maintenance *(Muller and Gaertner, 2004)*. La séquence de ces cinq activités est répétée plusieurs fois afin de développer progressivement le système. LOT est une méthodologie utilisée pour le développement d'ontologies *(Poveda-Villalón, 2012)*. Cette méthodologie se concentre sur (1) la réutilisation d'éléments (classes, propriétés et attributs) existants dans des vocabulaires ou ontologies déjà publiées et (2) la publication de l'ontologie, selon les principes du Web de données liées. Elle réutilise certaines activités d'ingénierie des connaissances définies dans la méthodologie NeOn *(Suárez-Figueroa*

*et al., 2015)*. Cette méthodologie définit les itérations sur les quatre activités suivantes : (1) spécification des exigences ontologiques, (2) implémentation de l'ontologie, (3) publication de l'ontologie et (4) maintenance de l'ontologie. La figure 63 illustre la méthodologie pour développer ce système d'aide à l'irrigation. Cette méthodologie permet à combiner le développement d'un système et le développement de l'ontologie utilisée par ce système. De plus, le résultat du système deviennent un outil précieux pour valider l'ontologie.

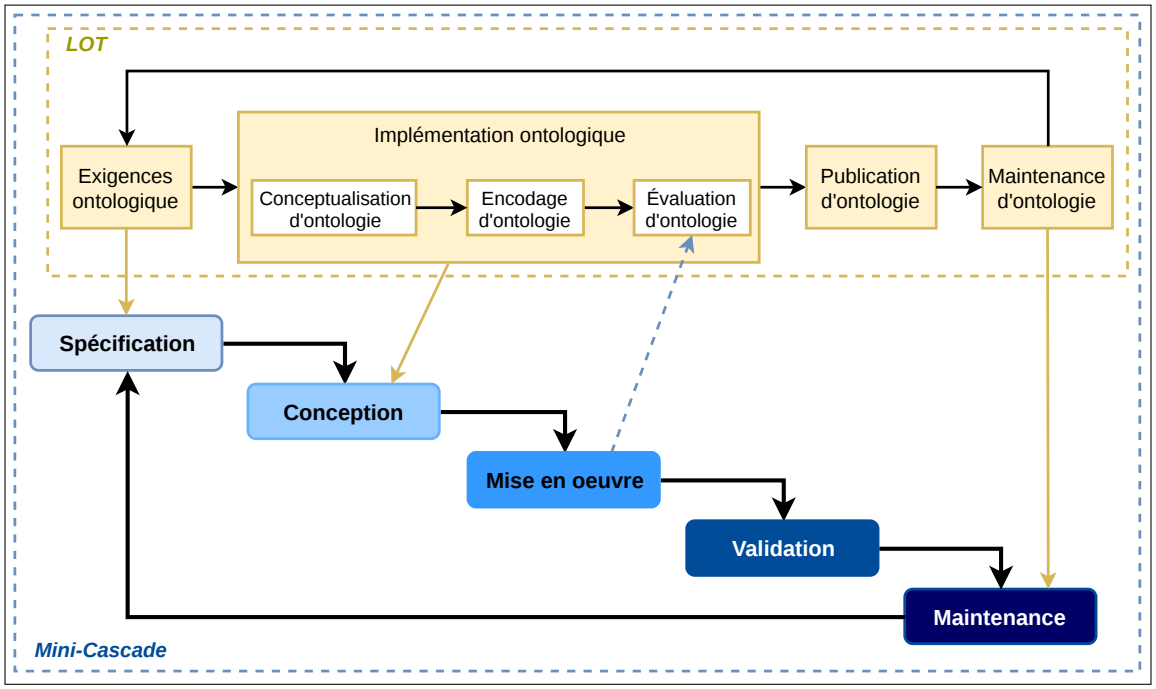



**Figure 63**: La méthodologie combinée de midi-cascade et LOT

Dans la première activité, la spécification de système, quatre sources d'information sont considérées et étudiées: (1) la méthode IRRINOV®, (2) un ensemble des données d'Arvalis, (3) un carnet de terrain et (4) les connaissances des experts de Arvalis ou INRAE. Notez que les données d'Arvalis ont été enregistrées sur une expérimentation agricole réelle réalisée par des agriculteurs dans une parcelle de maïs à Gaillac en France. Ces données sont utilisées plus tard pour la validation du système d'irrigation. Le résultat de cette activité est un document de spécification et un document des questions de compétences. D'une part, la spécification permet aux développeurs de système de comprendre le système. D'autre part, l'ensemble des questions de compétences aide à déterminer les besoins en vocabulaire de l'ontologie ciblée.

Dans la deuxième activité, la conception du système, il faut tout d'abord déterminer les entités et les données dans le système d'irrigation. Ils sont classifiés en

flux opérationnels. Chacun de ces flux est décrit par un phénomène observé (sol, pluie, culture) et ses propriétés associées : (1) l'humidité du sol, (2) la quantité de pluie, (3) le stade de développement de la culture et (4) le besoin en eau de la culture. La figure 64 illustre quatre flux opérationnels du système contextuel dédié à l'irrigation. Les entités de chaque flux et leurs états sont définis. Pour décrire les changements d'état, ce projet utilise les diagrammes états-transitions.

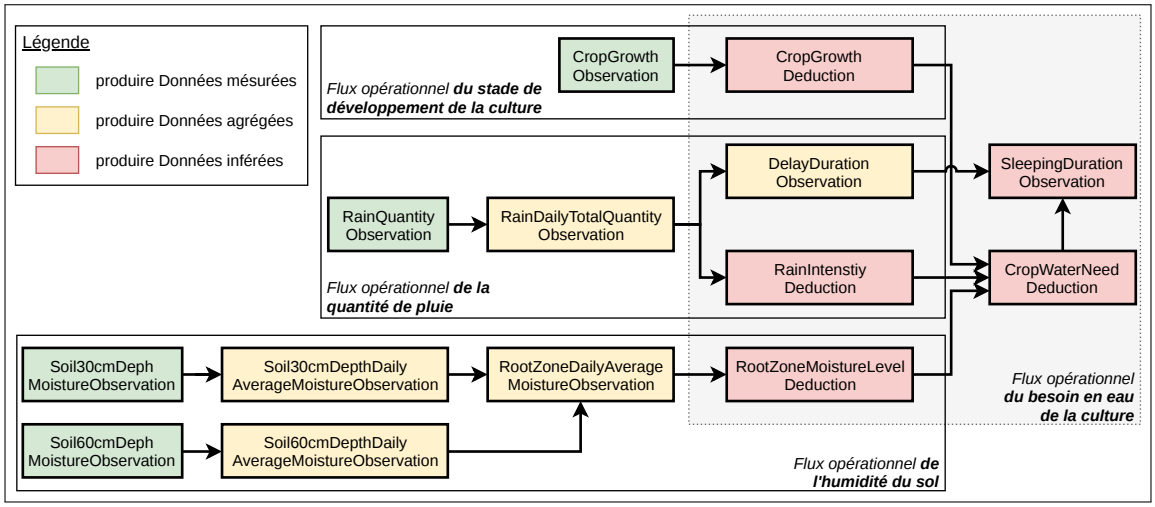



**Figure 64**: Quatre flux opérationnels du système contextuel dédié à l'irrigation

Ce projet propose une nouvelle ontologie intitulée IRRIG. Cette ontologie est une spécialisation de l'ontologie CASO pour l'irrigation. Elle étend le vocabulaire de CASO avec 31 nouvelles classes, quatre nouvelles propriétés, 71 individus et 24 règles. La figure 65 illustre l'ontologie IRRIG. Avec IRRIG, il est possible de modéliser les observations, agrégations, déductions et leurs données concernant l'irrigation plus précisément. De plus, les 24 règles sont un composant important pour raisonner sur les données.

Dans la troisième activité, la mise en oeuvre de système, ce projet propose un logiciel intitulé Ontogen qui joue le rôle du système d'aide à la décision dédié à l'irrigation. Ce logiciel écrit en Python et en Java, est développé en utilisant les services de la pile de services pour les systèmes contextuels. Notez que le système d'aide à la décision est seulement une partie du système contextuel d'irrigation de TSCF, donc Ontogen se compose de seulement six services : annotation, stockage, récupération de données, fusion, raisonnement et mise à jour du stockage. Dans ce logiciel, les 24 règles pour le service de raisonnement sont encodés en SWRL.

Dans la quatrième activité, la validation de système, ce projet utilise trois méthodes d'examinations : (1) tests unitaires, (2) tests d'échantillons et (3) test système. Le principe de tests unitaires est de créer des valeurs limites pour vérifier

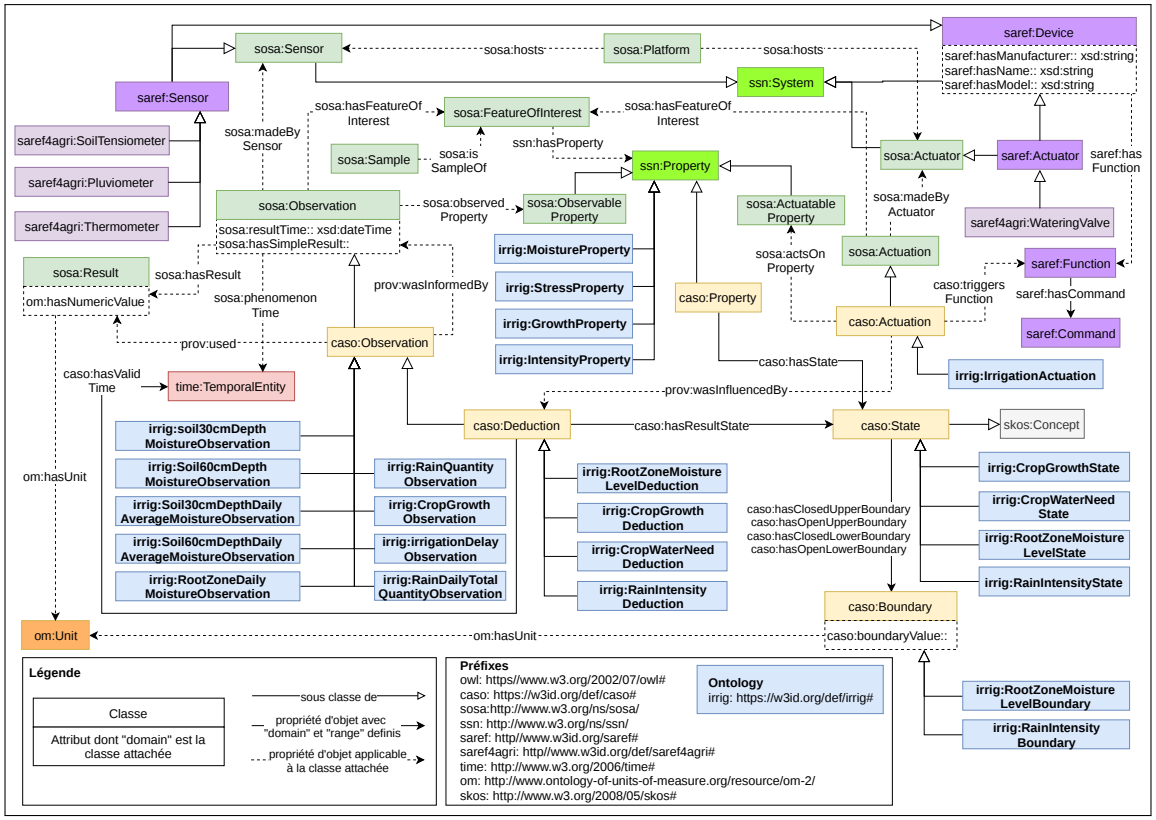



**Figure 65**: L'ontologie IRRIG

si les règles fonctionnent correctement. Dans les tests d'échantillons, on choisit par hasard est de choisir par hasard des valeurs d'un flux dans l'ensemble des données d'Arvalis pour voir si le système d'aide à la décision donne un résultat identique aux données enregistrées par des agriculteurs. Le principe d'un test système est de laisser le système consommer tous les données d'Arvalis, pour comparer la décision du système d'irrigation avec la décision des agriculteurs.

Dans la cinquième activité, la maintenance du système, ce projet utilise deux approches. Premièrement, l'équipe de développement du système joue aussi le rôle de maintenir et mettre à jour ce système. Deuxièmement, pour maintenir l'ontologie IRRIG, celle-ci est publiée sur GitHub, ce qui permet à l'équipe de développement de l'ontologie de maintenir et mettre à jour l'ontologie en s'appuyant sur les commentaires des autres utilisateurs.

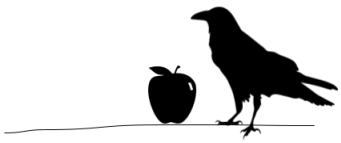
## Conclusion

Les trois contributions de cette thèse sont dans le domaine de l'agriculture 4.0. La première améliore l'évolutivité et la deuxième améliore l'interopérabilité du système contextuel. La troisième contribution est non seulement un exemple d'application

de ces deux premières contributions dans un vrai cas d'étude, mais également la transformation d'une méthode manuelle d'irrigation en un système d'aide à l'irrigation automatique. Même si les ressources pour former et évaluer les deux premières contributions sont dans le cadre de la e-agriculture, elles sont généralisées pour servir dans les domaines de l'éco-système de l'Internet ds Objets tels que les villes intelligentes ou les transports intelligents.

Toutes les contributions de cette thèse sont appliquées tout d'abord dans un système contextuel d'irrigation située dans TechnoHall, un lieu pour faire des expérimentations technologiques de l'INRAE. Ensuite, l'équipe TSCF plévoit la mise en oeuvre ce système contextuel dans l'AgroTechnoPôle à Montoldre, en France, pour faire des expérimentations réelles. À côté de cette partie pratique, quatre publications ont été produites à partir des travaux de cette thèse *(Poveda-Villalón et al., 2018a,b; Nguyen et al., 2020b,a)*.

Au moins quatre perspectives ont été identifiées pour faire suite à cette thèse. Premièrement, une perspective est d'améliorer la pile de services pour les systèmes contextuels : il est possible de créer une plateforme à partir de l'architecture de pile de services en développeant un protocole de communication entre les services. Une deuxième perspective est d'améliorer l'ontologie CASO : il est possible d'ajouter un vocabulaire relatif aux réseaux informatiques pour les systèmes contextuels. Une troisième perspective est le passage d'un système contextuel simple à un système contextuel adaptatif : mettre en place le service de reconfiguration permettrait d'avoir un système contextuel adaptatif. Enfin, une quatrièmement perspective est d'améliorer la méthode IRRINOV® : il est possible d'utiliser une technique d'apprentissage pour couvrir les situations pour lesquelles cette méthode n'a pas de réponse.



# VITA

**Quang-Duy NGUYEN** was born in 1989, in Hoàn-Kiém district, Hanoi, Vietnam. He attended local public schools and graduated from Chu Van An national high school in 2007. In 2012, he completed his Computer Engineering's degree at Hanoi University of Science and Technology (HUST). Later, he started a master dual degree program for computer science organized by the Institute Francophone International (IFI). He obtained a master's degree in Information Technology from the Vietnam National University (VNU), and a master's degree in Communication Systems and Networking (Master SIR) from the University of Claude Bernard Lyon 1, in 2016. He started his doctoral program in 2017 at the Technologies and Information Systems for Agrosystems (TSCF) research unit of the National Research Institute for Agriculture, Food and Environment (INRAE), in Clermont-Ferrand, under the supervision of Dr. Jean-Pierre CHANET (INRAE), Dr. Catherine ROUSSEY (INRAE) and Dr. Christophe DE VAULX (LIMOS).

In work, **Quang-Duy NGUYEN** develops capabilities both in engineering and science. He was educated to solve problems in multi-disciplinary domains in the boundary of information technologies, including embedded devices, communication networks, information systems, and artificial intelligence. He interests in building complex systems using Wireless Sensor Network, Ontology, and Reasoning system. In research, he has so far experience working in three research projects and contributing to six publications. The four publications relate to his doctorate are as follows.


[1] Maria Poveda-Villalón, **Quang-Duy Nguyen**, Catherine Roussey, Christophe de Vaulx and Jean-Pierre Chanet, "Besoins ontologiques d'un système d'irrigation intelligent: Comparaison entre SSN et SAREF", le 29es journées francophones d'Ingénierie des Connaissances IC 2018, 07/2018.

[2] Maria Poveda-Villalón, **Quang-Duy Nguyen**, Catherine Roussey, Christophe de Vaulx and Jean-Pierre Chanet, "Ontological Requirement Specification for Smart Irrigation Systems: A SOSA/SSN and SAREF Comparison," the 9th International Semantic Sensor Networks Workshop, ISWC 2018, 10/2018.

[3] **Quang-Duy Nguyen**, Catherine Roussey, Maria Poveda-Villalón, Christophe de Vaulx and Jean-Pierre Chanet, "Development Experience of a Context-Aware System for Smart Irrigation using CASO and IRRIG ontologies," MDPI Applied Science, 03/2020.

[4] **Quang-Duy Nguyen**, Catherine Roussey, Maria Poveda-Villalón, Christophe de Vaulx, Jean-Pierre Chanet and Camille Noûs, "CASO et IRRIG deux ontologies pour le développement de systèmes contextuels : cas d'usage sur l'automatisation de l'irrigation", 31es journées francophones d'Ingénierie des Connaissances IC 2020, 06/2020.