Deriving Service-Oriented Dynamic Product Lines Knowledge from Informal User-Requirements: AI Based Approach

Najla Maâlaoui, Raoudha Beltaifa, Lamia Labed Jilani

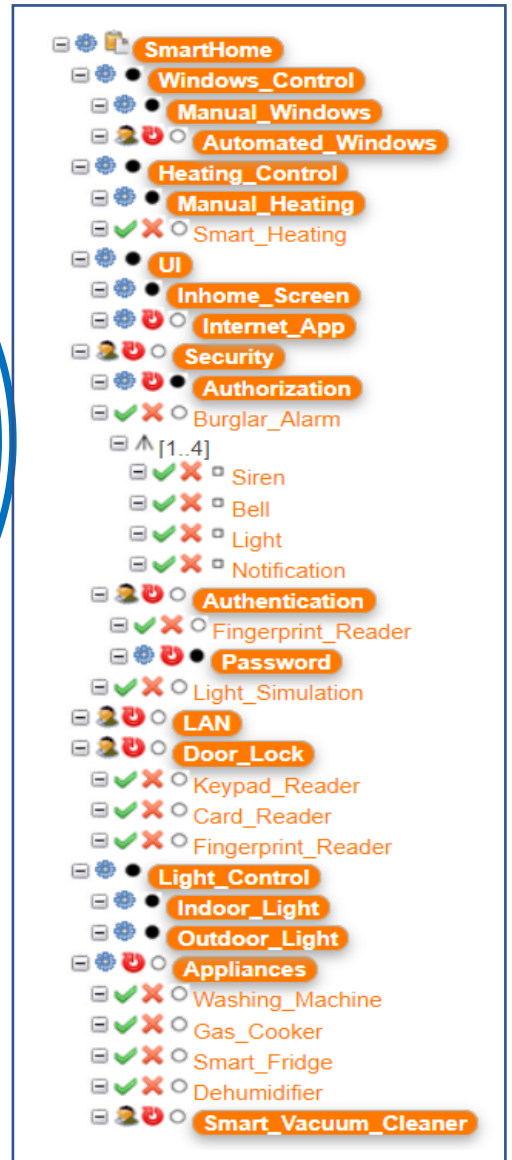
The Seventeenth International Conference on Software Engineering Advances

Outline

- Introduction
- Issues
- Proposed approach
- Expriment and Evaluation
- Conclusion and future directions

General context: Software product lines

Smart home options catalogue



Customer1

- ✔ SmartHome
- ✔ Automated_Windows
- ✔ Smart_Heating
- ✔ Security
- ✔ Burglar_Alarm
- ✔ Siren
- ✔ Light
- ✔ Authentication
- ✔ Appliances
- ✔ Smart_Fridge
- ✘ Dehumidifier
- ✔ Smart_Vacuum_Cleaner

Customer2

- ✔ SmartHome
- ✔ Smart_Heating
- ✔ Security
- ✔ Burglar_Alarm
- ✔ Light
- ✔ Siren

Customer3

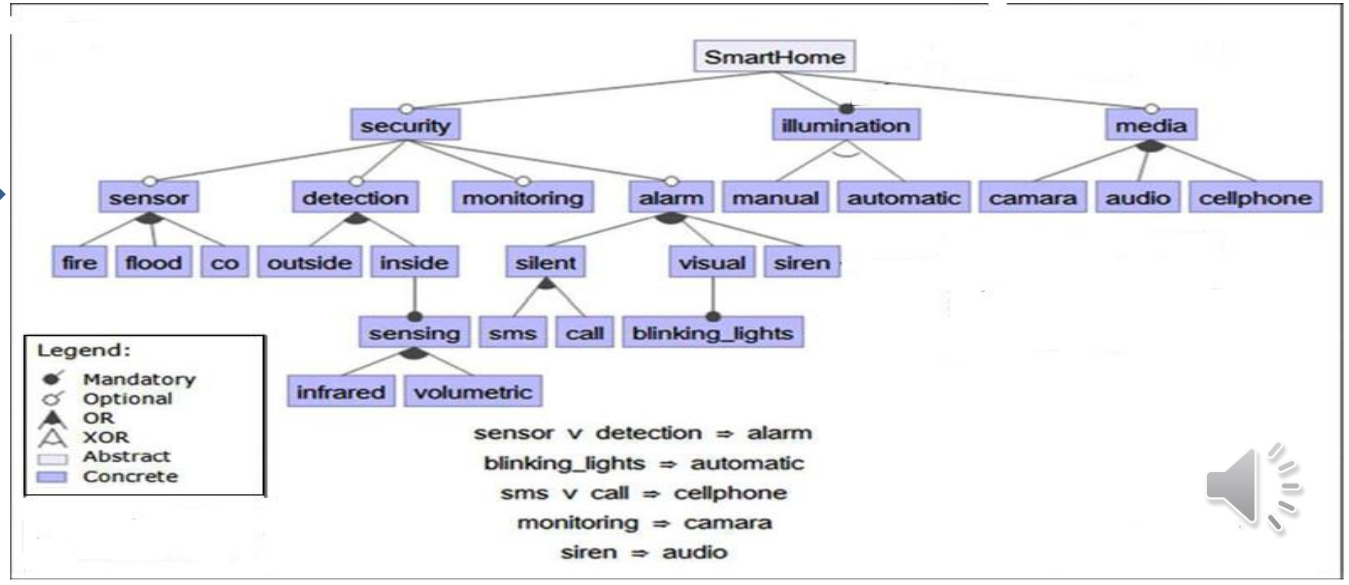
- ✔ SmartHome
- ✔ Automated_Windows
- ✔ Security
- ✔ Burglar_Alarm
- ✔ Notification
- ✔ Authentication
- ✔ Fingerprint_Reader
- ✔ Door_Lock
- ✔ Fingerprint_Reader
- ✔ Appliances
- ✔ Washing_Machine
- ✔ Gas_Cooker
- ✔ Smart_Vacuum_Cleaner

General context: Software product lines

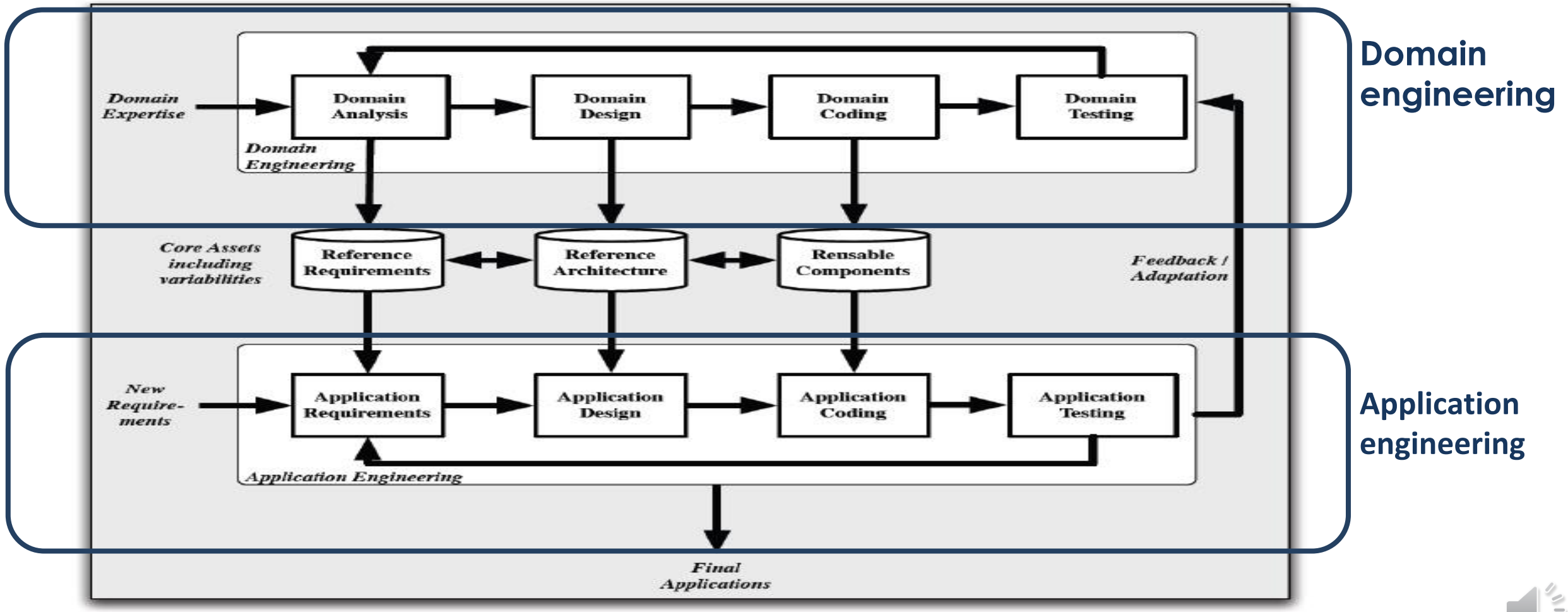
A **Software Product Line** is a **set** of software-intensive systems sharing a **common, managed set of features** that satisfy the specific needs of a **particular market segment or mission** and that are developed from a **common set of core assets** in a **prescribed way.** [Günter Böckle et al. (2005)]

Product 1	Product 2	Product 3
✔ SmartHome	✔ SmartHome	✔ SmartHome
✔ Automated_Windows	✔ Smart_Heating	✔ Automated_Windows
✔ Smart_Heating	✔ Security	✔ Security
✔ Security	✔ Burglar_Alarm	✔ Burglar_Alarm
✔ Burglar_Alarm	✔ Light	✔ Notification
✔ Siren	✔ Siren	✔ Authentication
✔ Light		✔ Fingerprint_Reader
✔ Authentication		✔ Door_Lock
✔ Appliances		✔ Fingerprint_Reader
✔ Smart_Fridge		✔ Appliances
✘ Dehumidifier		✔ Washing_Machine
✔ Smart_Vacuum_Cleaner		✔ Gas_Cooker
		✔ Smart_Vacuum_Cleaner

Smart home SPL feature model



General context: Software product lines



Günter Böckle et al. (2005)

General context: Service oriented Dynamic Software product lines

Technology evolution 1

Context analysis 2

Functionalities adaptation 3

Dynamic Software product lines

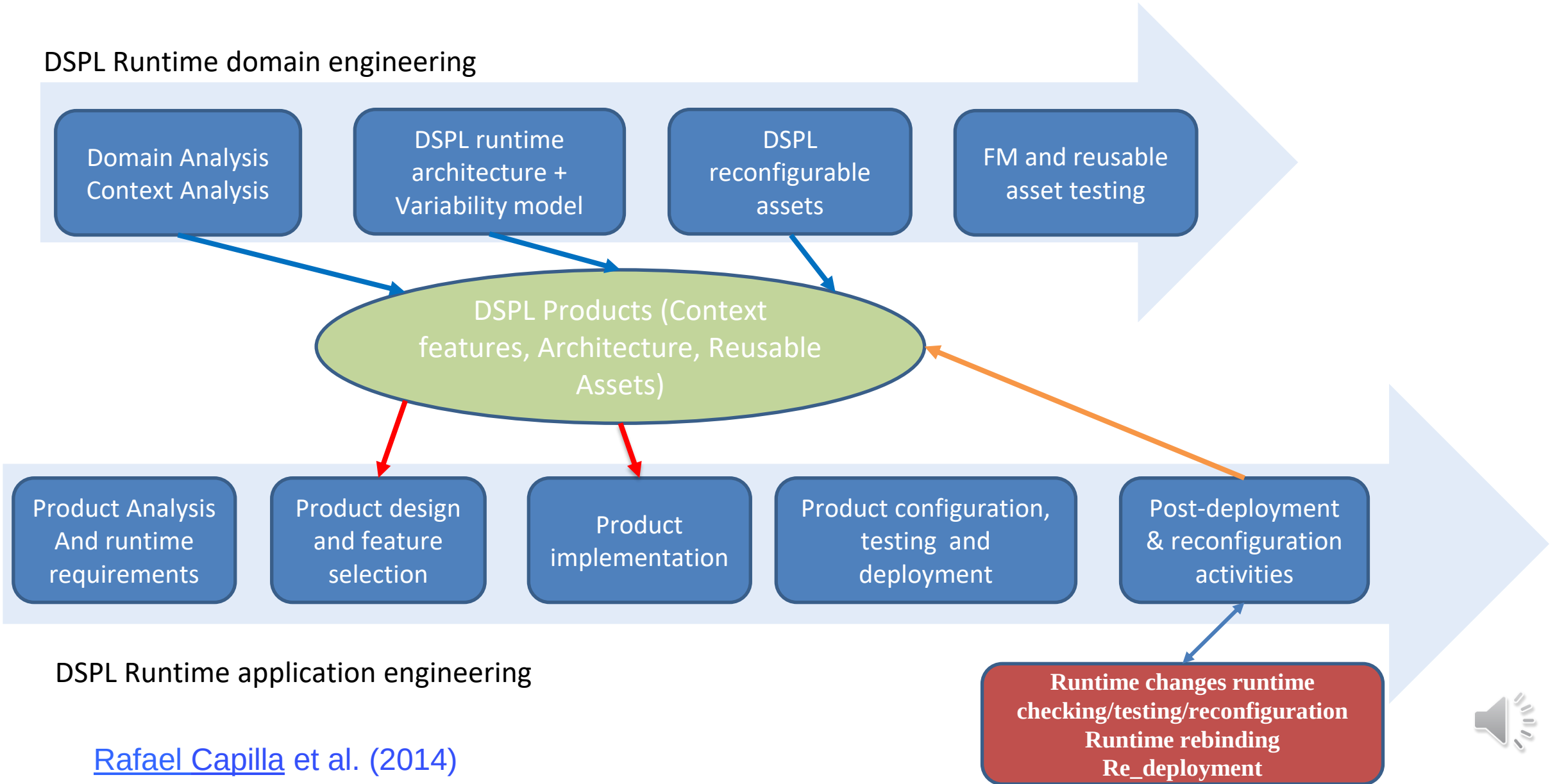
Service- oriented Dynamic Software product lines(SO-DSPL)

4 Dynamic execution

5 Requirements & context change

6 Self adaptation

General context: Service oriented Dynamic Software product lines



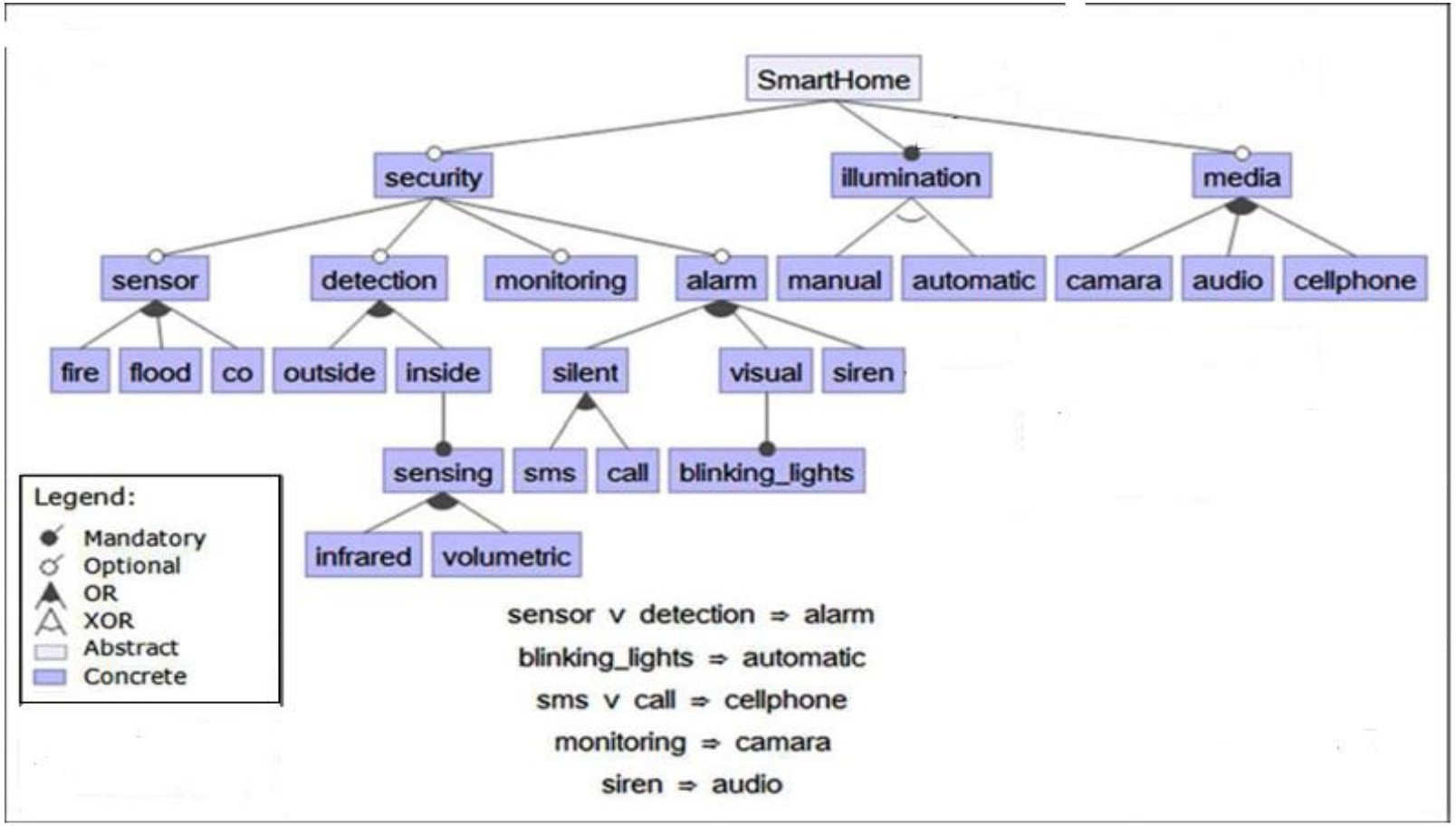
Rafael Capilla et al. (2014)

General context: Service oriented Dynamic Software product lines

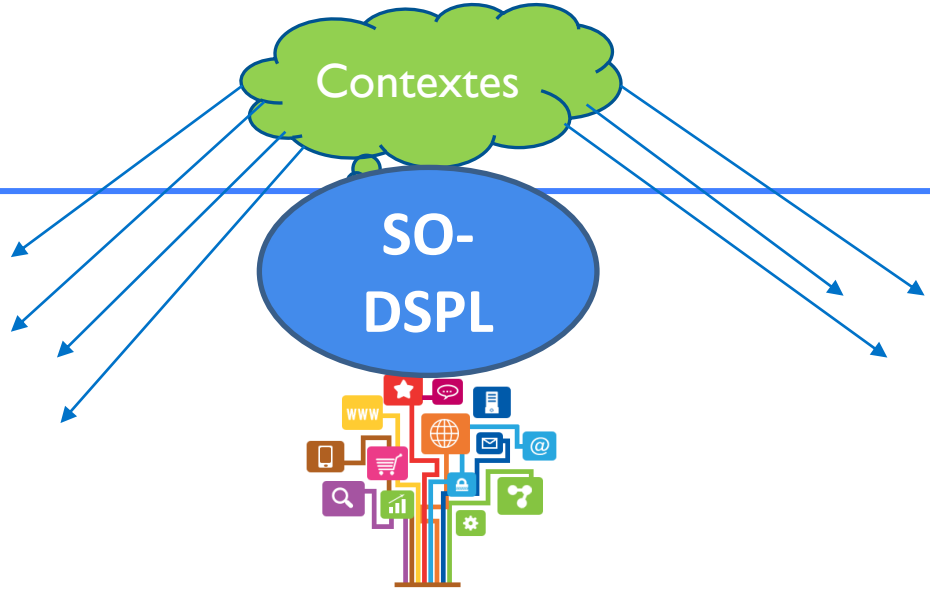
DSPL

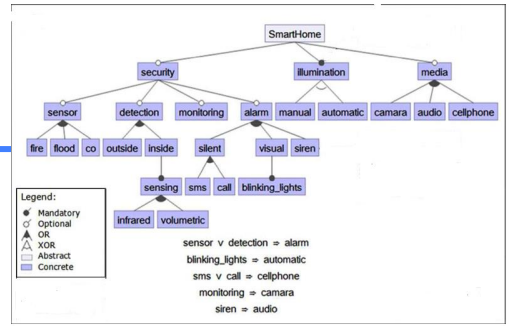
SOA

S0-DSPL



Issues





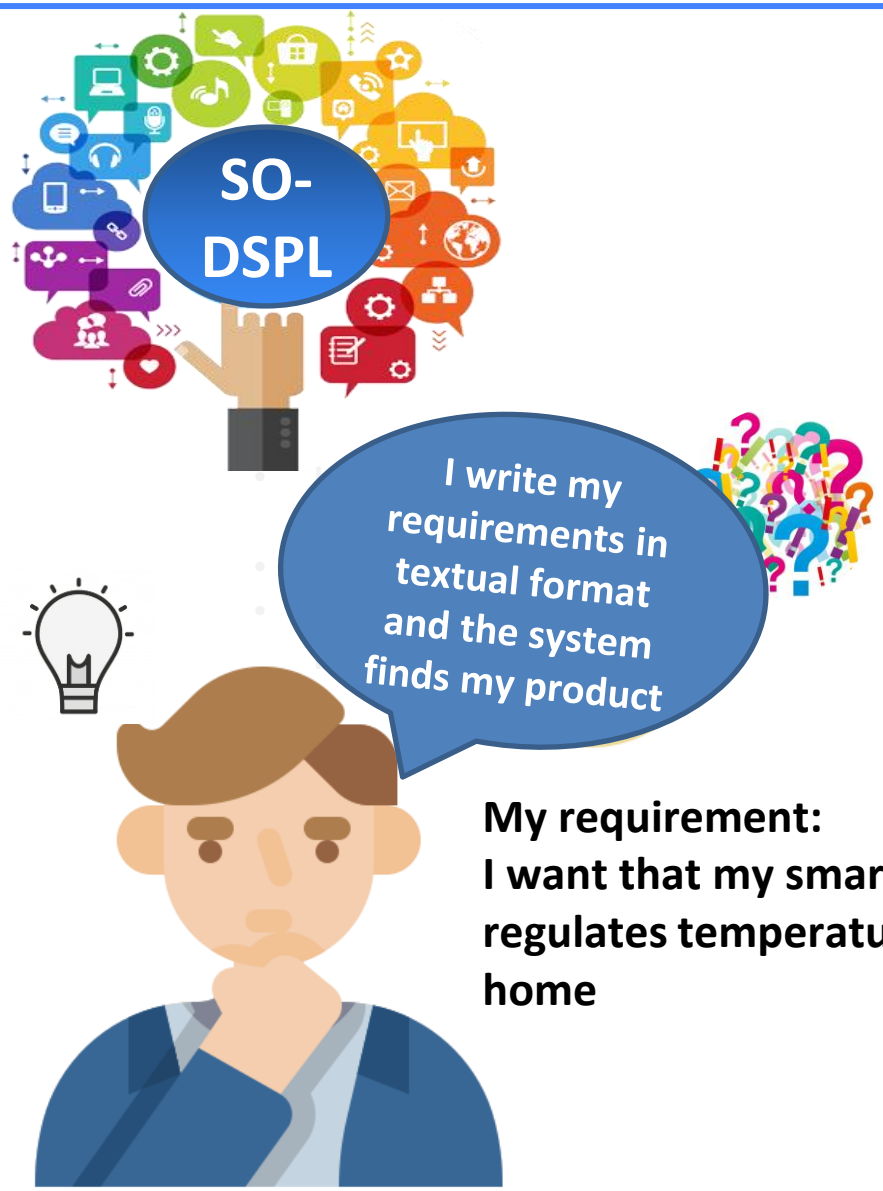
How can i make the best combination of options ?

How i configure?

I don't understand the mining of the options

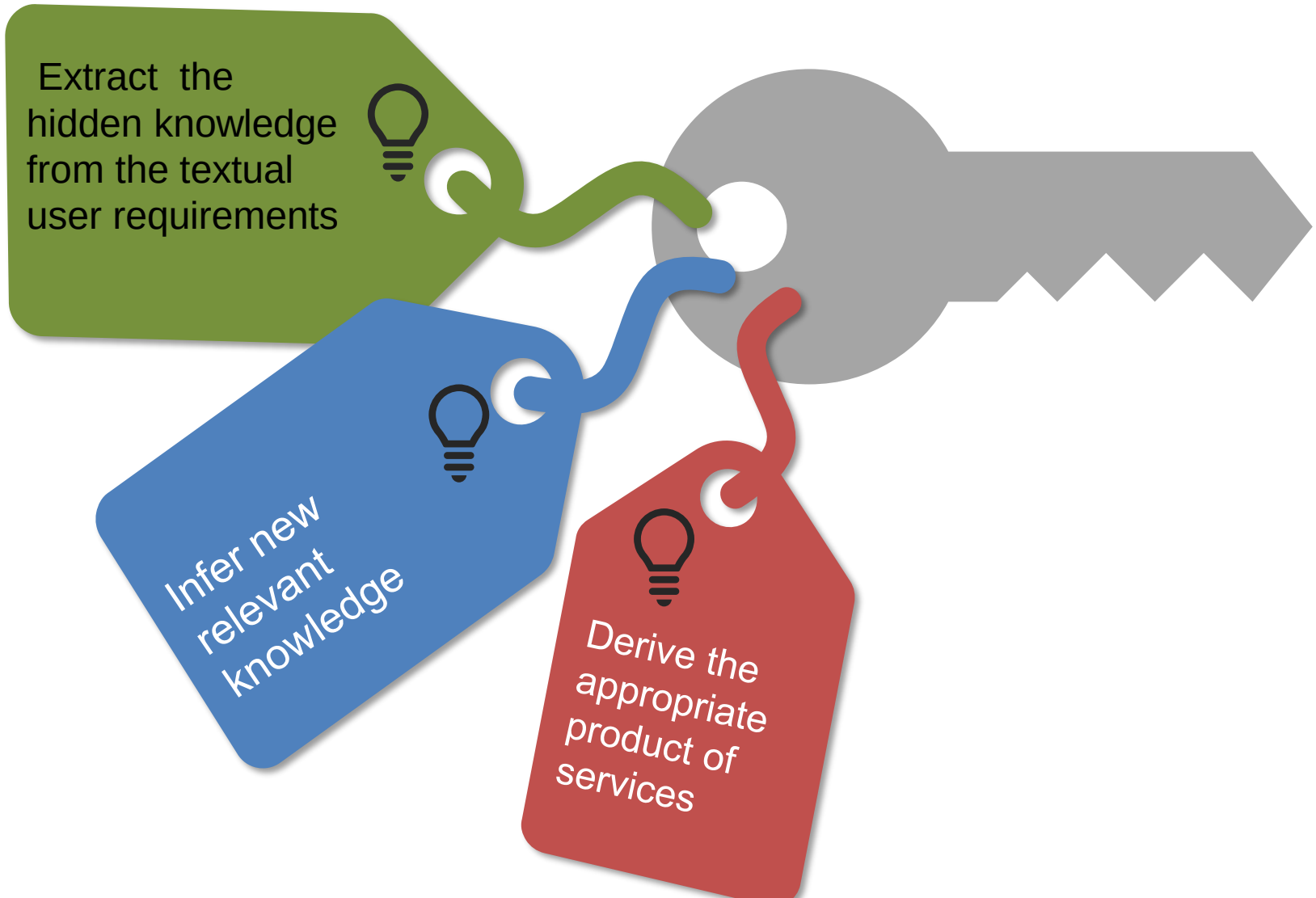
How can I describe the context to configure the adaptable product

Issues

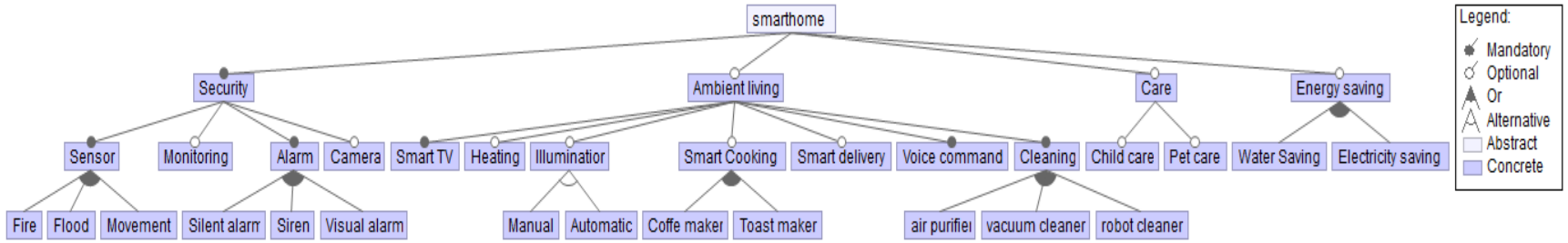


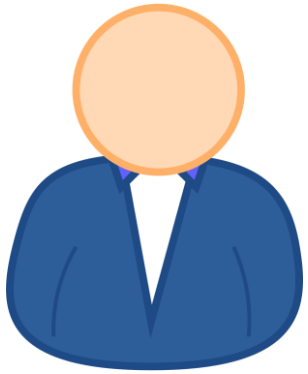
My requirement:
I want that my smart home automatically regulates temperature when i am at home

Solution: Deriving Service-Oriented Dynamic Product Lines Knowledge from Informal User-Requirements



Solution: Deriving Service-Oriented Dynamic Product Lines Knowledge from Informal User-Requirements





In the morning, I am very lazy so I think that my smart home shall have voice command feature to operate activities from bathrooms which accelerates my morning routines. Because my children Lora and Alex are always doing something stupid, my home shall have a vacuum cleaner to clean my home floors when anything spills. Sometimes, I am very tired since I work all the day, so I prefer that the smart delivery of my home may be able to order delivery food by simple voice command.

A core requirements have the following structure: <feature>+ <obligationdegree>? <goal>+ <item >* <condition>*

Where :

– * : denotes a zero repetition to an infinite number of times repetitions.
– + : denotes repetition once or more number of times.
– ? :denotes a repetition zero or one time.
– — :denotes a disjunction (it signifies OR).
– ! : denotes a negation

Elements	Core Requirement1	Core Requirement2	Core Requirement3
Feature	Voice command	Vacuum cleaner	Smart delivery
Obligation degree	must		should
Goal	operate	clean	order
Item	Activities from bathrooms	floors	food
Condition		When anything spills	Voice command

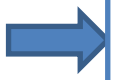
Solution: Deriving Service-Oriented Dynamic Product Lines Knowledge from Informal User-Requirements

Elements	Core Requirement1	Core Requirement2	Core Requirement3
Feature	Voice command	Vacuum cleaner	Smart delivery
Obligation degree	must		should
Goal	operate	clean	order
Item	Activities from bathrooms	floors	food
Condition		When anything spills	Voice command

A core requirements have the following structure:
<feature>+ <obligationdegree>? <goal>+ <item >* <condition>*

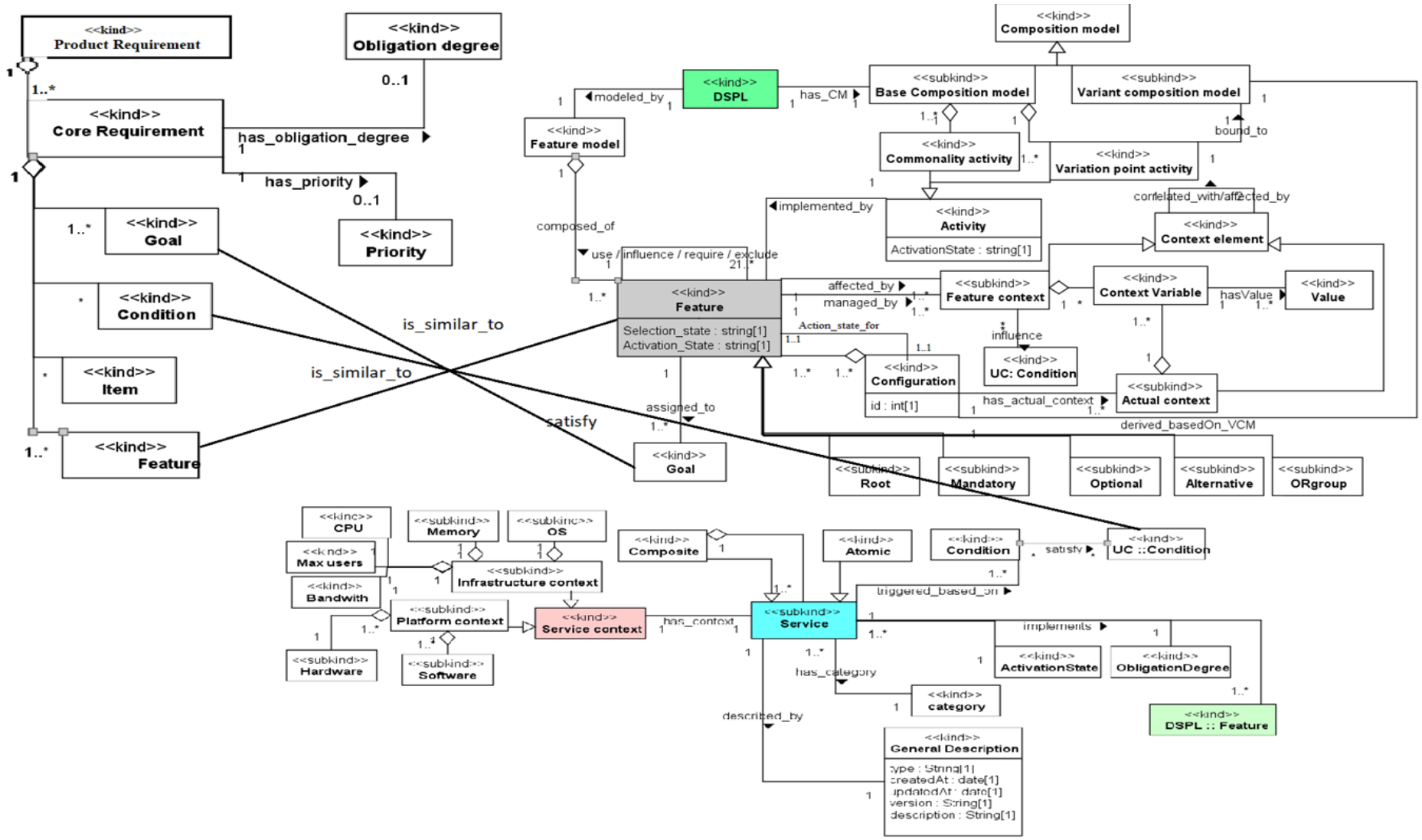
Where :
– * : denotes a zero repetition to an infinite number of times repetitions.
– + : denotes repetition once or more number of times.
– ? :denotes a repetition zero or one time.
– — :denotes a disjunction (it signifies OR).
– ! : denotes a negation

Product1 = [Sensor, Fire, Alarm, Visual alarm, Smart TV, Voice command, cleaning, vacuum cleaner, smart delivery, air purify, Care, child care]
Product2 = [Sensor, Fire, Alarm, Visual alarm, Smart TV, Voice command, cleaning, vacuum cleaner, vacuum cleaner, air purify]
Product3 = [Sensor, Fire, Flood, Alarm, Siren, Smart TV, Voice command, cleaning, vacuum cleaner, robot cleaner, smart delivery, air purify]

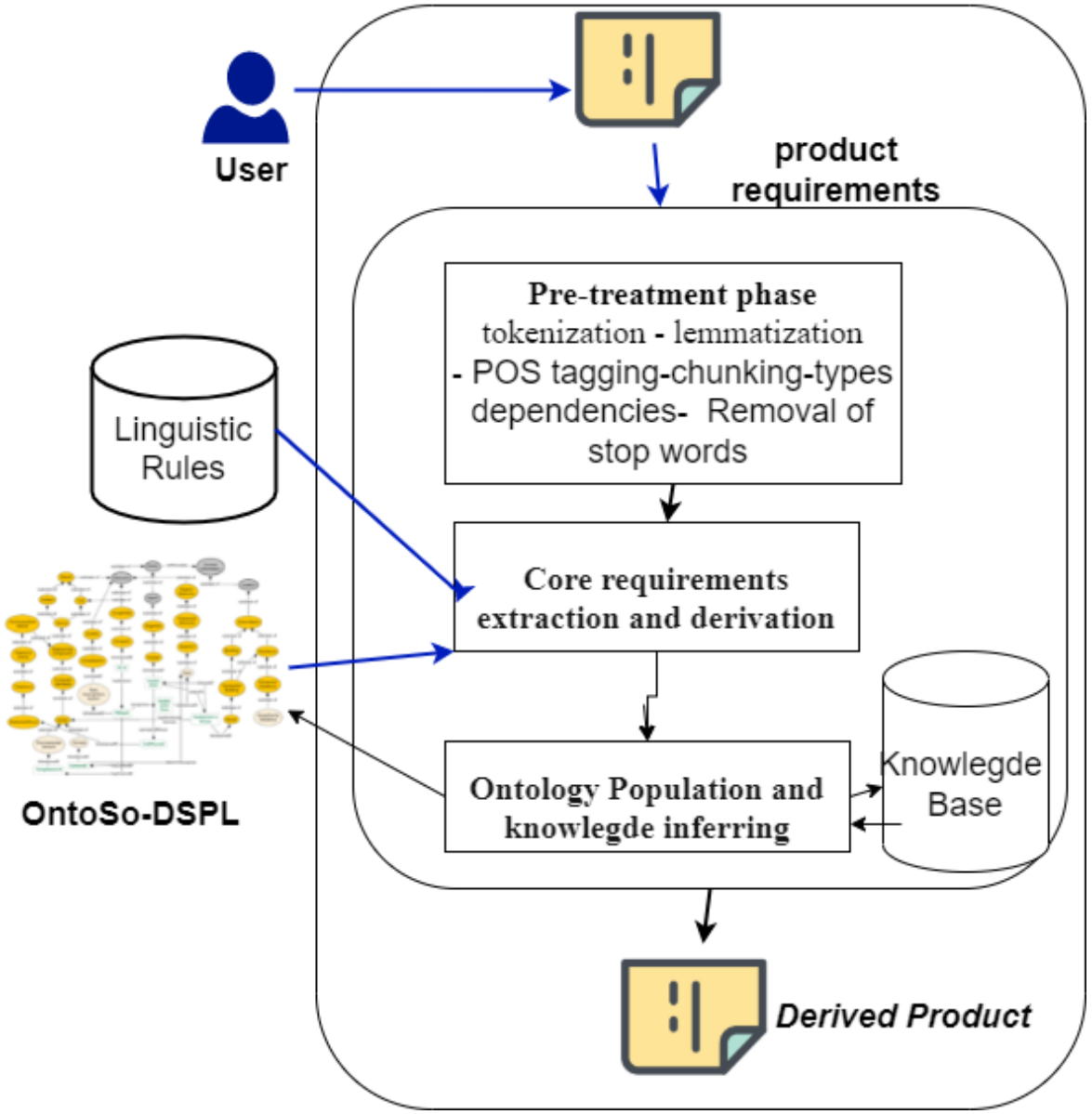


SO-Product1 = [Fire WS35, S fire, S Movement, Visual alarm WS10, Smart TV WS13,
S voice, Voice command, WS1101, vacuum cleaner WS125, smart delivery WS22, air purify WS45, child care WS234]
SO-Product2 = [S fire, Fire WS35, S Movement, Visual alarm WS10, Smart TV WS13,S voice, Voice command, WS1101, vacuum cleaner WS125,S Dirt detect, air purify,WS136]
SO-Product3 = [Fire WS35, S fire, S Movement, S flood, Flood WS321, Alarm WS1, Siren WS132, Smart TV WS13, S voice, Voice command WS110, vacuum cleaner WS125, robot cleaner WS127, smart delivery WS22, S Dirt detect, air purify WS136]

Solution: SO-DSPL Knowledge Extraction Framework



Solution: SO-DSPL Knowledge Extraction Framework



SO-DSPL Knowledge Extraction Framework : Pre-treatment phase

User requirements: My sensor should detect movement. if I get up late night, lights in my house should turn on to enhance convenience and safety

Chunks	POS tag	Types dependencies
(ROOT (S (NP (PRPMy)($NN sensor$)) (VP (MD should) (VP (VB detect) (NP (NN movement)))) (. .))) (ROOT (S (SBAR (IN If) (S (NP (PRP i)) (VP (VBP get) (PRT (RP up)) (NP (NP (JJ late) (NN night) (NNS lights)) (PP (IN in) (NP (PRPmy)($NN house$))))))) (VP (MD should) (VP (VB turn) (PRT (RP on)) (S (VP (TO to) (VP (VB enhance) (NP (NN convenience) (CC and) (NN safety)))))))	nmod:poss(sensor-2, My-1) nsubj(detect-4, sensor-2) aux(detect-4, should-3) root(ROOT-0, detect-4) obj(detect-4, movement-5) mark(get-3, If-1) nsubj(get-3, i-2) csubj(turn-12, get-3) compound:prt(get-3, up-4) amod(lights-7, late-5) compound(lights-7, night-6) obj(get-3, lights-7) case(house-10, in-8) nmod:poss(house-10, my-9) nmod(lights-7, house-10) aux(turn-12, should-11) root(ROOT-0, turn-12) compound:prt(turn-12, on-13) mark(enhance-15, to-14) xcomp(turn-12, enhance-15) obj(enhance-15, convenience-16) cc(safety-18, and-17) conj(convenience-16, safety-18)	nmod:poss(sensor-2, My-1) nsubj(detect-4, sensor-2) aux(detect-4, should-3) root(ROOT-0, detect-4) obj(detect-4, movement-5) mark(get-3, If-1) nsubj(get-3, i-2) csubj(turn-12, get-3) compound:prt(get-3, up-4) amod(lights-7, late-5) compound(lights-7, night-6) obj(get-3, lights-7) case(house-10, in-8) nmod:poss(house-10, my-9) nmod(lights-7, house-10) aux(turn-12, should-11) root(ROOT-0, turn-12) compound:prt(turn-12, on-13) mark(enhance-15, to-14) xcomp(turn-12, enhance-15) obj(enhance-15, convenience-16) cc(safety-18, and-17) conj(convenience-16, safety-18)

SO-DSPL Knowledge Extraction Framework : Core requirements extraction and derivation

1) To associate each token of the chunks results to its accorded Core requirement form element, we have compiled a list of linguistic rules that cover the most possible cases.

Each rule refers to:

- the grammatical category of the token
- its linguistic environment that is the series of units that precede and follow it
- its typed dependencies with the other tokens.

By applying the suitable rule for the token, we can associate it with the corresponding core requirement element and then, a textual core requirement will be derived based on its corresponding pattern.

Rule	Principal Pos	Dependency type antecedent	Grammar antecedent	Description
R1	NN	Nsubj(t2, t1)	(*PRP*?|JJ*)| (*DT*?|JJ*) NN(*MD*|VB| CC)	If the token t1 is a noun (NN), it has a dependency nsubj with a verb (t2) and the antecedent is true then the t1 is a feature
R2	NN	Conj(t1,t2)	(*CC*|,) (*PRP*?|JJ*)| (*DT*?|JJ*) NN(*MD*|VB)	If the token t1 is a noun (NN), it has a dependency Conj with a verb (t2) and antecedent is true then the t1 is a feature
R3	VB	Nsubj(t1,t2)	MD?TO? *VB*|VBN	If the token t1 is a verb (*VB*|VBN), it has a dependency nsubj with a noun (NN) the antecedent is true then t1 is a goal
R4	VB	Nsubj(t1,t2)	MD? TO? *VB*|VBN |VBP RP	If the token t1 is a verb (*VB*|VBN |VBP), it have a dependency nsubj with a noun (NN), it is by a token t2 having the type "RP" and the antecedent is true then the concatenation of t1 and t2 is a goal
R5	MD	Aux(t2,t1)	MD VB	If the token t1 is a modal Verb (MD),its followed by a verb, it has an aux dependency with a verb the antecedent is true then t1 is an obligation degree
R6	(*NN*|NNS)	Obj(t2,t1)	VB(*PRP*?|JJ*)| (*DT*?|JJ*)NN	If the token t1 is a noun (NN), it is included in a NP chunk,it has an obj dependency with a verb the and antecedent is true then t1 is an item
R7	NN	Nsubj(t2, t1)	(*PRP*?|JJ*)| (*DT*?|JJ*)NN (MD?) (*VB*?|VBZ)VBN (IN)	If the token t1 is a noun (NN), it has a dependency nsubj with a verb (t2) and the antecedent is true then the t1 is an Item
R8	(*PRP*|NN |NNS)	Nsubj(t1,t2)	IN (*PRP*?|JJ*)| (*DT*?|JJ*) *PRP*|NN|NNS (*VB*|CC)	If the token t1 is a noun or a possessive pronoun (PRP),it is proceeded by an "IN" that is equals to if/ when/where, it has a Nsubj dependency with a verb and the antecedent is true then t1 is a Subject of a condition

SO-DSPL Knowledge Extraction Framework : Core requirements extraction and derivation

- A certainty factor is assigned to each rule serving later as a degree of membership to the core requirement element attributed with the rule.
- A certainty factor is an assigned weight to each rule. This weight can be interpreted as an evaluation measure of a rule of its

This factor represents **uncertainty**

- The certainty factor is greater when there is a close relation between the antecedents and the consequence
- A fixed minimum threshold of belief in the truth of a rule at 30 % (CF=0.3) is used to select the relevant rules after trying different thresholds in the interval of [0.2 ..0.5] to fix the threshold that maximize the precision and the recall of our proposed approach.
- All the rules with a CF below this threshold are deleted because they deal with particular cases and helps to improve the error rate of the system.

SO-DSPL Knowledge Extraction Framework : Ontology Population and Knowledge inferring

- To populate the ontology with the new core requirements instance
- To derive new knowledge based on the extracted core requirements and knowledge that already exists to conceptualize the running SO-DSP

List of core requirement components

Ontology Population and Knowledge inferring engine

Populated ontology with new knowledge

Derived product

Selected services

SO-DSPL Knowledge Extraction Framework : Ontology Population and Knowledge inferring

SWRL RULES	DESCRIPTION
Requirement(?r) ^ swrlb:contains (?r, ``Shall ")-> has_priority(?r,Essential)	If the requirement contains the term ‘shall’ then its priority is ‘essential’
Requirement(?r) ^ swrlb:contains (?r,`` Should ")-> has_priority(?r,Recommended):	If the requirement contains the term ‘Should’ then its priority is ‘Recommended’
Goal(?g) ^Feature(?f) ^assigned_to(?f,?g) ^characterized_by(?g,Dynamic_evolution)->has_variability_type(?f,Dynamic):	If the goal is assigned to a feature and characterized by a dynamic evolution then the feature has a dynamic variability
Requirement(?r) ^ Goal(?g)^ Feature(?f) ^satisfy(?g,r) ^assigned_to(?f,?g) ->selection_state (?f,`` selected ")	If a goal satisfy a user requirement then the feature assigned to the requirement has a "selected" state

Evaluation

- We have implemented a tool that supports our proposed approach steps including the linguistic rules.
- OWL (Web Ontology Language) is used to populate the proposed SO-DSPL ontology using data from the input client's product requirements.
- Smart home requirements data set is used as an application use case
- To evaluate the performance of our approach, the existed dataset is augmented using data Augmentation algorithms which consists in altering an existing data to create a new one.
- Recall and precision are user to evaluate our approach

	TP	FP	FN	Precision	Recall
DataSet	55400	6241	4261	89.81%	92.85%

Conclusion

SO-DSPL Knowledge Extraction Framework :

- analyses and understands automatically the user requirement.
- extracts user requirements and builds a requirement in accordance with the core requirement structure
- based on a set of linguistic rules and the support of uncertainty.
- relevant knowledge are inferred and relevant services are selected to derive the entire user product
- Automatically made the entire process of the extraction and population approach
- Uses semantic by the use of the ontology reasoning capabilities

Future directions

- Enrich our approach with linguistic rules that allow the extraction of context changes.
- Combine this work with recommender system that will derive the appropriate services for the generated requirement.

Questions?

Najla Maalaoui

National school of computer science

Najla.maalaoui@ensi-uma.tn