IDFML: A Meta Language for Heterogeneous DSMLs Coordination

Naima Essadi, Adil Anwar

Abstract*: *The popularization of both Software Language Engineering (SLE) and Model Driven Engineering (MDE) as well as the increasing complexity of systems induce new implementation practices. Indeed, many teams of experts collaborate to implement a same system. Every team uses her own domain specific modeling language (DSML) to represent her concerns. Consequently, by the end of the modelling phase, we get many heterogeneous models elaborated using different DSMLs. These models need to be composed to get a whole view of systems, to be able to validate and simulate behaviors. However, many recent researches choose to compose modelling languages rather than models themselves, but until now there are no standard or generic techniques for that. Although, MDE and SLE provide tools and concepts for modeling, customizing and processing business concepts as single activities, in contrast they provide little support for coordinating between these activities. In this paper, authors propose an interface-based approach to coordinate DSMLS in order to compose and coordinate their respective models. They began by giving generic guidelines of DSMLs specification and composition aspects. Then, they introduce IDFML a Meta language for defining both DSMLs interfaces and coordination. Actually, the proposed Meta language gives a metamodeling background to coordination which enables to benefit from MDE tools and techniques. They finally demonstrate the applicability of the approach using a Connected Indoor Transport Service System to conclude by summarizing benefits of the proposed Methodology

Keywords*: Domain Specific Language, *Abstract Syntax, Concrete Syntax, Coordination, Interface

I. INTRODUCTION

Models are first artifacts of MDE and as models gain progressively ground over objects, they became day after day primary artifact for software development too. The story began when OMG initiates Model Driven Architecture (MDA) [10] to separate both business and technology concerns. The MDE came then to give a concrete translation to MDA principles. In fact, MDE provides a plethora of concepts and tools to conceive models and manipulate them. Consequently, the need of modelling growth could not anymore be satisfied by UML the famous general-purpose modelling language. Indeed, the vulgarization of software language engineering was at the rendezvous by providing new capabilities to create DSMLs.

A DSML is specifically tailored for a domain. It enables to express concerns using domain vocabulary and notations. Actually, using a DSML rather than UML or any other general-purpose modelling language improves both communication between stakeholders and products quality. Moreover, the improvement of communication prevents conception misunderstanding which impacts positively implementation costs. Actually, domain expert understand clearly models elaborated with their own vocabulary and concepts and consequently, Models could also be done by experts themselves. In fact, DSMLs reduced the gap between problems and solutions scopes as both of them are now expressed by the same language.

However, the use of a specific modelling language for every business domain caused an accidental heterogeneity due to various modelling languages used by teams collaborating to implement a system. Actually, the complexity of nowadays systems and software induces the implication of many teams of experts as every team uses a different modelling language. Subsequently, by the end of the modelling phase, we get many heterogeneous models representing different views of a same system.

We believe that the ability to compose heterogeneous DSMLs is a key solution to compose and coordinate resulting heterogeneous models. The composition of these models is absolutely needed for many reasons: to have global view of systems, to be able to perform analysis of scattered information over models and also to ensure consistency, maintenance and evolution of systems.

Many recent works investigated composing DSMLs issue but most of them had as purpose the definition and reuse of modeling languages. In contrast, this work is concerned by DSMLs composition to be able to compose and coordinate their conforming models.

Furthermore, the use of interface concept to compose DSMLs have been discussed by many relevant works. However, there are still many challenges to overcome.

In this paper, we try to answer following questions:

How can we define a DSML interface?

How can an interface be self-contained? Hide the complexity and implementation or other specifications that aren't relevant for outside?

How can we link interfaces of different DSMLs?

How can we use interfaces defined at language level to compose and coordinate heterogeneous models?

The reminder of this paper is organized as follows. We first gave an overview of DSMLs specification as well as languages workbenches in Section 2. In Section 3 we introduce our approach. Then, section 4 illustrates the applicability of the approach in the case of a Connected Indoor Transport Service System. Finally, in Section 5 we discuss related work to conclude the paper in section 6.

Revised Manuscript Received on November 01, 2020.

***** Correspondence Author

Naima Essadi*, Siweb, E3S, EMI, Mohammed V University in Rabat, Rabat, Morocco. Email: naimaessadi@gmail.com

Adil Anwar, Siweb, E3S, EMI, Mohammed V University in Rabat, Rabat, Morocco. Email: anwar@emi.ac.ma

II. DSML SPECIFICATION

A DSML is first of all a language which can be defined using classical formalisms like context-free grammar especially BNF. However, to benefit from MDE emergence, DSMLs was first designed with UML profiles and finally using meta-models. Doing so, we give to DSMLs another dimension and momentum. DSMLs could then be manipulated as MDE concepts and all MDE tools and advances could be therefore applied to them.

A. White Box Specification

A DSML is usually defined by its white box specification the five-tuple L = {A, C, S, Ms, Mc} [6,7].

The abstract Syntax "A" is the most important part of the specification. It gives all DSML's concepts by describing vocabulary and concepts of the language and how they may be combined to create models [8]. Actually, the abstract syntax defines concepts, their relationships and their well-formedness rules. Moreover, it is unique for a DSML and assures a pivotal role between various concrete syntaxes and different forms to express semantics [1]. The Abstract Syntax is often described using OMF Class Diagram [16].

The concrete syntax "C" of a language is its body, considering the Abstract syntax as a soul [1]. Its role is to represent a model to our human senses [1]. Actually, it provides notation to represent and construct models [8]. A language can provide various concrete syntaxes for a same abstract syntax. However, two main shapes of concrete syntax exist: textual and visual, the textual form gives variables, objects and expressions, it is suitable to capture complex expressions and details. Unfortunately, textual form remains difficult to manage beyond a great number of lines. On the other side, visual syntax gives graphical icons representing views; this form allows expressing concepts in an intuitive and understandable way but it is not suitable to represent complex and detailed information.

A concrete syntax provides alphabet of basic symbols used and also scanning and parsing rules stating about syntactical constructs [1].

On the other hand, Semantics "S" gives a clear idea about what a language represents and means [8]. It communicates a subjective understanding of the linguistic utterances of a language [1]. Semantics must be expressed clearly to avoid misinterpretation. It should also provide rich ways of interacting with the language [8].

Semantics definition is optional and may take many forms [8]: Translational semantics, Operational semantics, Extensional semantics and Denotational Semantics.

Additionally, The Syntactic Mapping "Ms" defines relation between concrete and abstract syntaxes. It could be represented as a M2M transformation having as input the Concrete Syntax Model and as output the Abstract Syntax Model.

The last tuple element "Mc" represents the semantics Mapping that gives correspondences and relations between semantics and Abstract syntax elements. It relates and map element of abstract syntax to elements of semantic domain. It can also be represented as a transformation M2M having the Semantic Model as input and the Abstract Syntax Model as output. This transformation relates Semantic Model Element to Abstract Syntax Model Concepts. For Denotational semantic Model, semantic Data of Semantic Domain are related to Abstract Syntax Model concepts like data Types, entities and associations... However, for an operational Semantic Model, Processes and parameters are related to operations and entities of the Abstract Syntax Model.

B. Black Box Specification

A language could also be specified by its black box definition that hides its irrelevant details. This kind of definition is means of abstraction and a good solution to do that is interfaces.

A language Interface is a set of elements that are exposed to outside and thus are available for other languages. It is defined at language level and then holds for all models of the language [1]. Moreover, an interface hides implementation details to expose just needed information. Doing so, it first decreases languages complexity, it protects from unexpected evolution and it reduces dependency. Indeed, an interface is an exposed contract that promotes reuse and substitution as well as the interface contract is respected [9].

The Abstract Syntax could be considered as a simple definition of DSML Interface, in this case, all concepts of the language will be accessible for other languages [1]. Also, an excerpt of Abstract Syntax could be considered as Interface, this excerpt is used to reduce Abstract Syntax's complexity, and then, provides a more concise and simple part of languages.

Two different types of Interfaces could be defined: Required Interface and Provided Interface. A Required Interface of a language defines references and information needed by the language itself from other languages while a Provided Interface offers elements to be referenced by other languages. Moreover, the later one should contain references to self-Abstract Syntax elements provided in an identification scheme available for other languages [1]. The use of these references gives to other DSMLs ability to coordinate with the owner DSML, to reuse some part of it and to avoid redefinition of existing concepts. Furthermore, this coordination allows also concerns separation between languages to permit to every part fulfillment of its ultimate purpose.

In software engineering, a new era has followed the interface concept introduction, we believe it will be the same for DSMLs. Indeed, the definition of interface for DSMLs is considered as the key of modularization. Interfaces are very useful and could help to face various challenges that arise from current modelling practices.

In software Engineering, Interface concept is mature and commonly used. However, in Software Language Engineering this concept is still ambiguous, ad-hoc and consequently not widely used.

Many recent works proposed definitions for language interfaces. So far, there is no clear or formalized specification for that. Although there are many language workbenches that allow the DSML's specification as white boxes, they do not allow a black box specification for languages.

C. Language Workbenches

Both the increasing need of modeling and SLE advance cause a popularization of language definition which became no more reserved for few experts. Nowadays, everyone can create a new DSML to cover his concerns, and various language workbenches are available to such a point that we may hesitate about the tool to use. There are three kinds of language workbenches: Textual, Graphical and Projectional.
Textual workbenches allow definition of textual languages like Xtext [11], this kind of tools are grammar based and enable modeling complex concepts. Graphical tools on their side allow languages' visual definition of languages like the industrial tool MetaEdit+ [13] of MetaCase or academic tools like AToMPM [12] and GME [14]. These tools are more friendly hence suitable for domain experts. However, projectional tools [17] like JetBrains MPS [18] are more flexible and allow different forms of representation and modeling to be combined.
Although, these tools facilitate the creation of new languages, they offer little or almost no support to compose them.
In This paper, we use Sirius Framework [15] of Obeo Designer as language workbench. It is a graphical tool based on Ecore metamodel [16]. It enables the definition of the abstract syntax conforming to Ecore metamodel, the concrete syntax as well as the language graphical editor.

III. FACING MODELS HETEROGENEITY

The new modeling practices and the SLE tools advance make that at the end of modeling phase we get many heterogeneous models elaborated using heterogeneous DSMLs. These models need to be composed, integrated and coordinated for many reasons: to analyze information scattered over many models, to get a whole view of designed system in order to validate its features, to ensure its consistency and its evolution…
Many recent works investigated this issue and one of the important solutions is to coordinate DSMLs rather than models. Indeed, we believe that the coordination of DSMLs instead of every piece of model is a good optimized manner to resolve this issue. Actually, coordination done at language level is consequently hold by model level [1].

In this section, we gave some terminological precision before a detailed description of our approach.

A. Heterogeneous DSMLs Coordination

There are three levels of heterogeneity as previously described in our earlier works. However, this paper emphasizes language heterogeneity. Actually, by heterogeneous DSMLs, we mean languages with different metamodels but conform to the same Meta-metamodel, more precisely languages based on Ecore metamodel.

This paper is also about coordination of languages. The coordination is a form of composition where coordinated parts remain independent and unaltered.
Language coordination is therefore a form of weak composition that retains the independence of coordinated languages artifacts while linking them to guarantee a common objective.

B. Approach Description

Our approach proposes to coordinate heterogeneous DSMLs as black boxes. This form of definition hides implementation and concepts of languages to expose only relevant elements. Although, a black box definition hides the language complexity, it is self-contained and provide interfaces to be used as joining points between languages.

Fig.1 gives a high overview about of the proposed approach where DSMLs are exposing their interfaces that are linked to each other using coordination relationships. In our previous paper [19], we gave a list of some intended relationships between languages. We propose a DSML to describe a language interface. We call it IDFML (Interface Description for Modeling Languages).

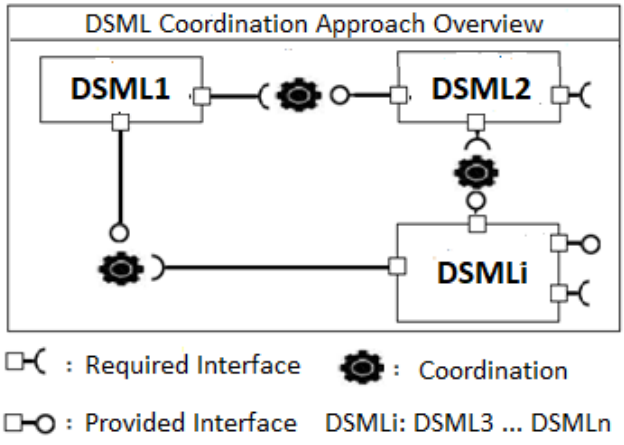

Fig. 1.DSML Coordination Approach Overview

Our approach includes three main steps: DSMLs' black-box specification, Coordination relationships definition and Models coordination. The two first steps are performed at language level whereas the last step concerns model level.

- DSMLs' black-box specification: this first step aims to use IDFML language to define DSMLs as black-boxes that expose interfaces as coordination points. The analysis of language's abstract syntax as well as language's main purpose help to identify both required and provided interfaces.
- Coordination relationships definition: The purpose of this step is to link interfaces using coordination relationships according to IDFML Meta language, metamodel in Fig.2 gives a non-exhaustive list of possible relations. The result of the first and second steps is an IDFML model conforms to metamodel of Fig.2 The model represents involved languages as black-boxes related using coordination relationships.
- Models coordination: the last step of our approach aims to coordinate heterogeneous models belonging to languages involved in earlier steps. The result of this last step is a coordination model conforms to the language model elaborated at first and second steps.

C. IDFML Meta-Language

The IDFML language allows the specification of a black box definition for DSMLs. It aims to define their Interfaces as well as composition relationships between them. The IDFML language provides following concepts:

- MetaPackage: concept representing a DSML. It contains a set of "Interface" elements.
- ProvidedInterface: concept representing a provided interface as a coordination point of a DSML. It has also a reference to the "MetaPackage" that it belongs to.
- RequiredInterface: concept representing a required interface as a coordination point of a DSML. It has also a reference to the "MetaPackage" that it belongs to.
- CompositionRelationShip: concept used to define a relationship between interface and core elements. This element is a super Meta Class of all possible composition relations to be hold between languages. The abstract syntax of Fig.2 lists some of them. However, a detailed definition of this relations has been given in [19].

The abstract syntax of IDFML displayed in Figure 2 is defined according to the coordination metamodel introduced in our previous works. The MetaPackage Meta element contains both "RequiredInterface" and "ProvideInterface" elements that are linked to each other using "CompositionRelationShip" Meta element. The composition relationship could be either a coordination or an integration relation.

IV. CASE STUDY

To demonstrate applicability of our approach, we applied it to a Connected Indoor Transport Service System (CITS). We use the IDFML language for modeling the composition at language level in order to be able to compose and coordinate heterogeneous models of the CTIS system.

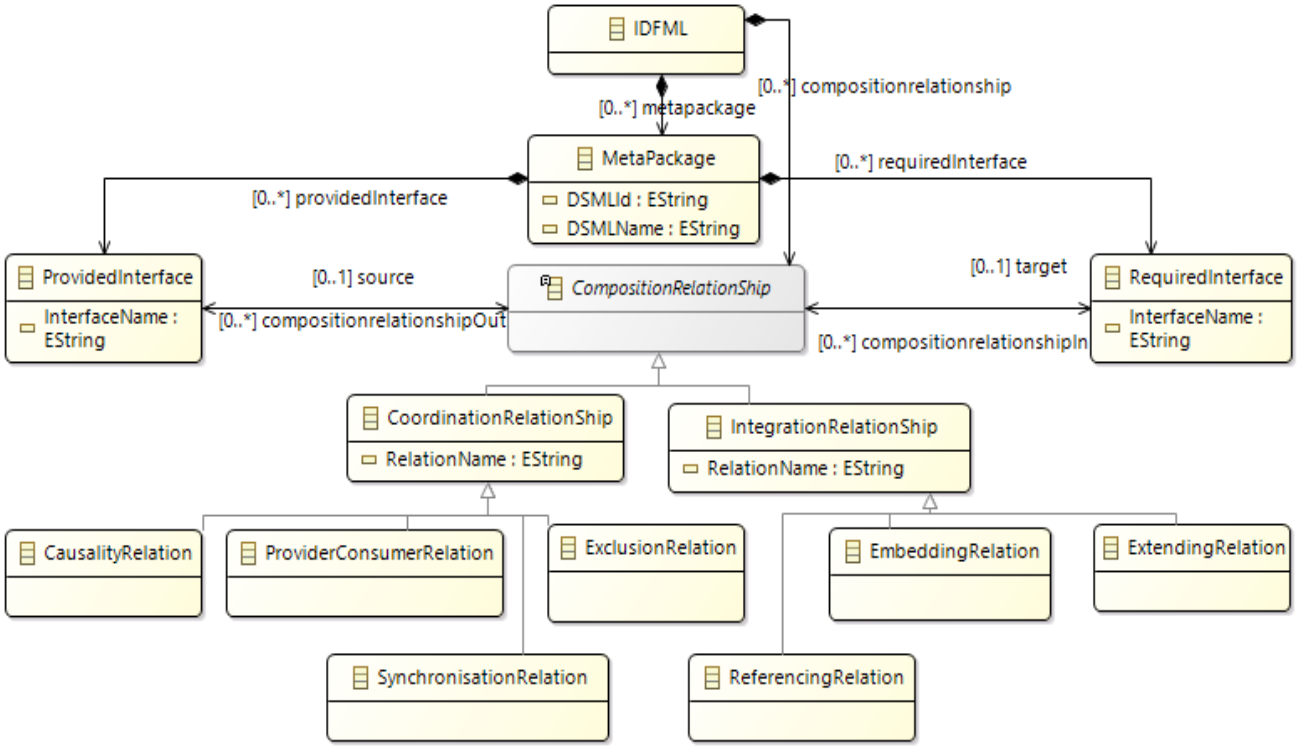

Fig. 2.Excerpt of IDFML Abstract Syntax

A. Connected Indoor Transport Service System

CITS system is composed by many parts. In this paper we are focusing just on two of them. The first part is the Indoor transport service activity which aims to transport items by robots from a source location to a target location according to assigned missions. The second part concerns IoT aspect of this system, actually robots are considered as connected objects.

B. Finite State Machine DSML

The DSML used to describe Robots work in the CITS System is Finite State Machine (FSM) DSML. Fig.3 gives the abstract syntax of this language described using Ecore meta-metamodel [16]., while Table. I describe a concrete syntax for the language. The root element of the metamodel is "FSM". It is the container of all elements of the abstract syntax. The "State" element represents the state of a robot during its transport service. A state can be initial, final or current. Moreover, as the robot's state changes according to received events, it has an outgoing and an incoming transition.

Table- I: FSM DSML Concrete Syntax

Concept	Concrete syntax
State	
Current State	
Transition	

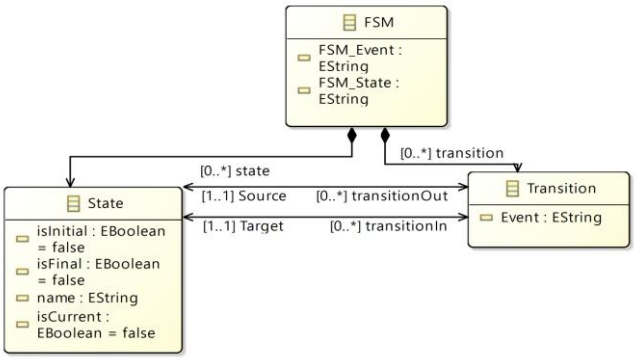

Fig. 3.Abstract Syntax of FSM DSML

Fig.4 represents a model conforms to FSM DSML. This model has been elaborated using Sirius Framework which enables us to create a graphical editor by defining the "fsm.odesign" view. The defined view is then assigned to our model to be editable in a user-friendly way. The Robot routine Model contains five states that describe Robot states during its transport service work. A Robot is in the sate "Ready" as initial state. Then, the event "MissionReceived" causes the change of Robot's state to "En Route for Provisioning" (ERP). At this state a Robot is moving to the source location to get the Item to transport. When the Robot arrive to the provisioning location its state became "Provisioning". After provisioning, the Robot moves to the Target location to deliver the item. When a robot reaches the delivery location, its state changes from "En Route for Delivery" (ERD) to "Delivering". Finally, the state return to Ready when delivery is finished.

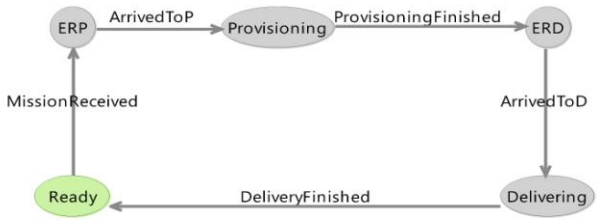

Fig. 4.Robot Routine FSM Model

C. ThingSee Purpose DSML

We use the DSML ThingSee Purpose (ThingSee) DSML of MetaCase to describe IoT concerns in the CITS System.

Fig.5 gives a metamodel excerpt of ThingSee DSML defined using Ecore meta-metamodel.

The ThingSee abstract syntax has "Purpose" as root element which contains all concepts of the syntax for instance "State", "Transition" and "Action". Every "State" has an incoming and an outgoing "Transition". On the other hand, a "Transition" is related to a sensor. The excerpt includes "LocationSensor", "BatterySensor" and "GeofenceSensor". The ThingSee DSML contains other sensors that we didn't include for simplification purpose and also because they are not relevant for our case study.

We also use the Sirius Framework to define the concrete syntax as given by table II.

Besides, models of Fig.6 and Fig.7 are conformed to ThingSee DSML. The first one describes battery check activity while the second one is about localization.

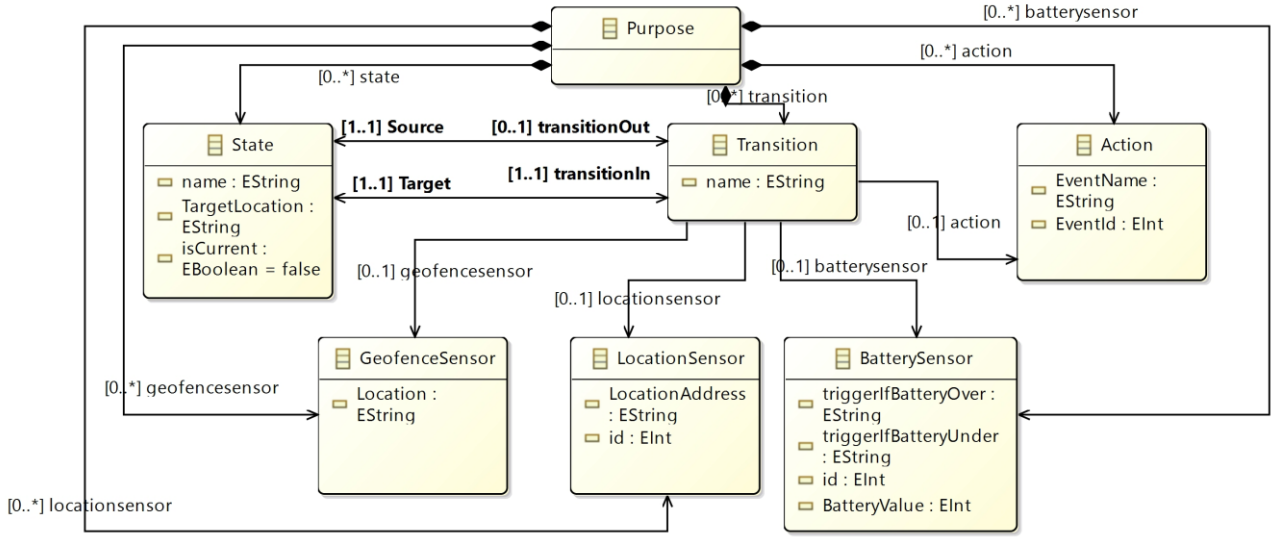

Fig. 5.Abstract Syntax of ThingSee DSML

Table- II: ThingSee DSML Concrete Syntax

Concept	Concrete syntax
State	
Action	
BatterySensor	
LocationSensor	

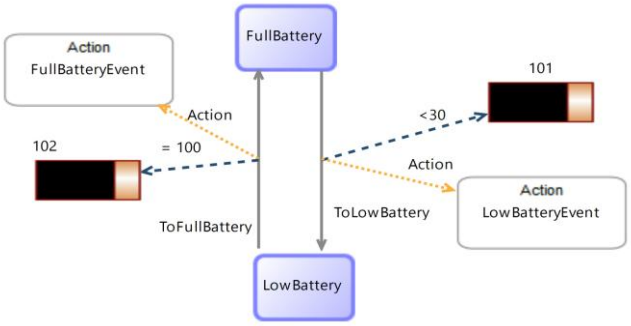

Fig. 6.Battery Check ThingSee Model

The battery check activity aims to watch battery level. If the threshold of observed element battery is under 30% then the Battery Sensor generates an event "LowBattery" and the state of the concerned element changes from "FullBattery" to "LowBattery". By the same way, the state returns to "FullBattery" when the battery threshold became 100%.

D. FSM and ThingSee DSMLs Coordination

We use the IDFML meta language to coordinate between FSM and ThingSee DSMLs.

The sequence Diagrams of Fig.8 describes CITS system behavioral. Actualy, in Fig.8 (a), the location sensor is activated when a robot is in "ERP" and "ERD" states. The sensor triggers the "arrivedEvent" when destination is reached. On his turn, the loading sensor is activated when robot is on "Provisioning" and in "Delivering" states, this sensor triggers respectively "ProvisioningFinished" and "DeliveryFinished" events. Besides, Fig.8 (b) describes Battery Sensor activity which is activated according to robot battery threshold. When a robot receives a "LowBattery" event it must wait for the "FullBattery" event to continue its routine.

The behavior of CITS system helps us to depict relevant interfaces to be defined for each language.

For the FSM DSML, we depict a provided interface and a required interface named respectively: "Fsm State ProvidedInterface" and "Fsm Event RequiredInterface".

However, we define two interfaces for the ThingSee DSML. The required one is "Thing See State RequiredInterface" and the provided one is "ThingSeeEventProvidedInterface".

To coordinate the two languages, we create a model conforming to IDFML metamodel displayed by Fig.9. This model has as MetaPackage instances FSM and ThingSee DSMLS. Every MetaPackage contains interfaces that are joined using coordination relationships.

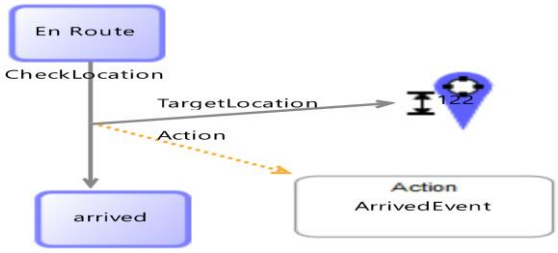

Fig. 7.Location Check ThingSee Model

E. CITS Sytem Models Coordination

The coordination achieved at language level holds to models too. Models we need to coordinate are ones of Fig.4, Fig.6 and Fig.7. The Fig.4 model is a inodel of MetaPackage FSM, where models of Fig.6 and Fig.7 are models of MetaPackage ThingSee.

Fig.11 illustrates a model of the IDFMLModel as well as the editor elaborated using Sirius Framework. Heterogeneous models of both DSMLs FSM and ThingSee are coordinated using the coordination relationships "ProviderConsumerRelation" and "CausalityRelations".

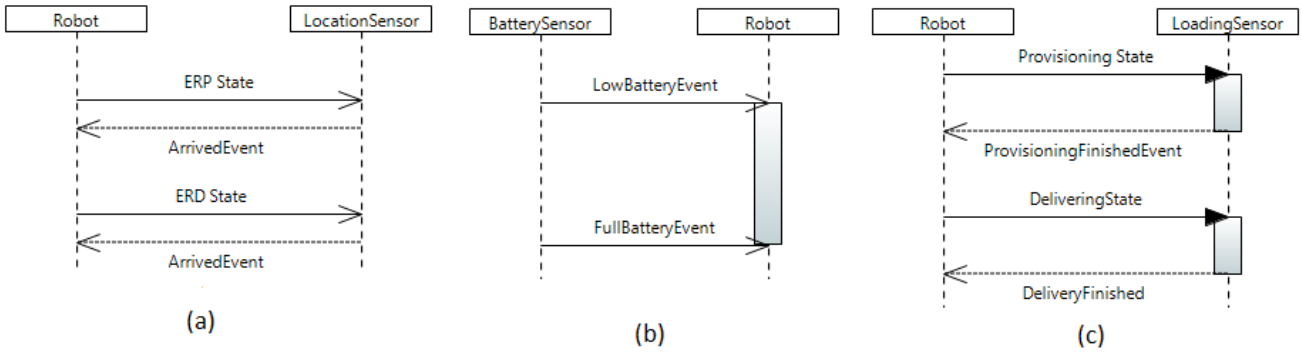

Fig. 8.CITS System Sequence Diagrams

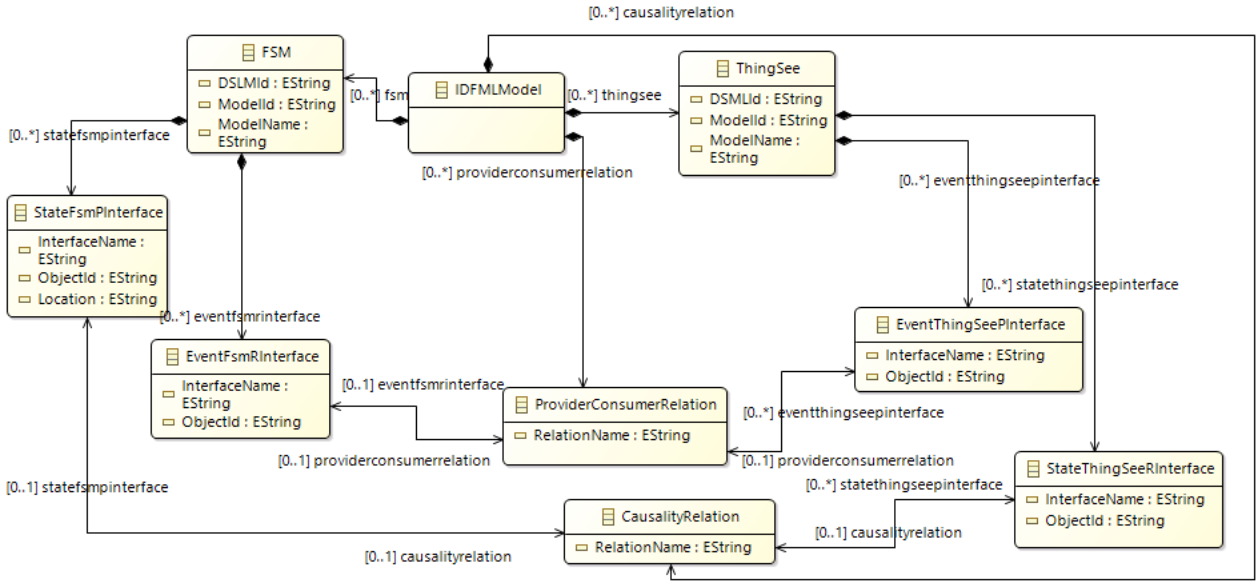

Fig. 9.IDFML Model for CITS System

Table III summarizes approach's concepts and CITS's instances according to OMG Meta levels [9]. The M3 level contains concepts defined by the IDFML Meta language's abstract syntax of Fig.2. The M2 level's elements belong to the IDFML Model of Fig.9, Whereas, the M1 level's elements are displayed by Fig.10 and are specific to the CITS system. It is important to notice that each level defines the one below and is conformed to the one above.

V. RESULT AND DISCUSSION

Several works have investigated black-box specification for metamodels and DSMLs. This type of specification allows modularization of languages and models. Authors in [2] proposed a toolset that enables to enhance metamodels with import and export interfaces. This proposition assures separation of concerns and information hiding. Thus, imported interfaces are considered as distinct elements and the defined components as self-contained units. In addition, 'Composite models' is a modularization technique introduced in [4] and its main idea is the export and import interfaces declaration that define model elements provided to and obtained from distributed models. In this technique an imported interface is assigned to an exported interface while an exported interface can serve several imported interfaces. Under this approach, defined interfaces hide the complexity of metamodels elements. Furthermore, authors in [3] use the same composite models 'technique as a formally modularization mechanism to address distributed models. Components are self-contained as all references links elements within the same component. They are at the same time interrelated using interfaces. Actually, this technique enables to hide information and allows local consistency checks. In [5] authors propose an approach based on interface and interface-base composition operators. This approach allows creating reusable self-contained black box meta-model component that could be composed flexibly and systematically. The export and import interfaces defined by previous works are similar to required and provided interfaces we propose. Indeed, like described approaches, we propose a black-box definition of languages that hides irrelevant elements for coordination. However, our approach defines a Meta language to specify that in a noninvasive way. The metamodeling aspect gives our approach the benefit to use MDE tools and concepts like transformations that allows automatization and artifacts generation. In addition, our approach informs about coordination relationships semantics as under composition relationship we put many other specific relations. Furthermore, our approach might be used for both structural and behavioral languages composition without modifying their abstract syntax.

Table- III: Concepts and Instances' Summary

Meta Level	Concept/Instance			
M3	MetaPackage	ProvidedInterface	RequiredInterface	CoordinationRelationship
M2	FSM DSML	StateFsmPInterface	EventFsmRInterface	ProviderConsumerRelation CausalityRelation
	ThingSee DSML	EventThingSeePInterface	StateThingSeeRInterface	
M1	Robot Routine Model	StateForSensor	EventRequiredInterface	ProviderConsumerRelation CausalityRelation
	BatteryCheck Model LocationCheck Model ProvisioningCheck Model DeliveryCheck Model	LowBatteryEvent FullBatteryEvent ArrivedEvent ProvisioningFinishedEvent DeliveryFinichedEvent	ER_RequiredState DLV_RequiredState Prov_RequiredState	

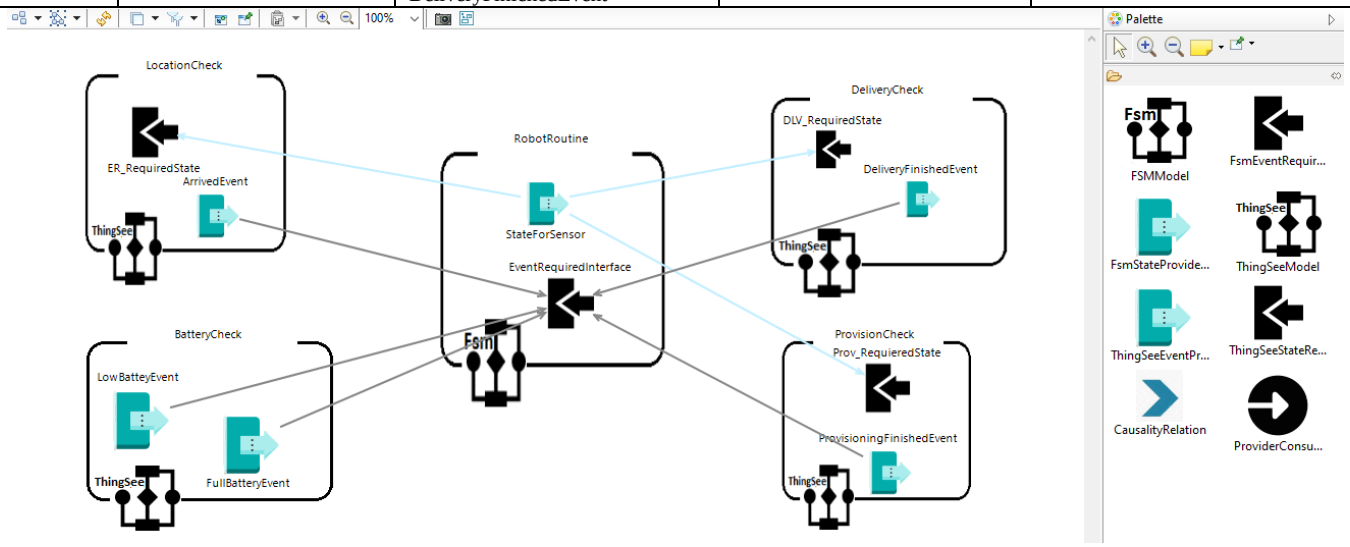

Fig. 10. CITS System Models Models Coordination Using IDFML Editor

VI. CONCLUSION AND FUTURE WORK

In this paper, we introduced an interface-based approach to coordinate between heterogeneous DSMLs. In fact, we propose the IDFML, a Meta language allowing a black-box specification for modeling languages. The black-box specification we propose hides languages complexity and exposes relevant parts that might be used as coordination points. Subsequently, the coordination done at language level enables coordination at model level. Furthermore, we demonstrated the usefulness of the approach by applying it to the Connected Indoor Transport Service System where known DSMLs are involved: the FSM and ThingSee DSMLs for instance. Actually, the IDFML language allows the definition of a coordination model to achieve coordination between heterogeneous models of both languages. Actually, our proposition gives a metamodeling background of both language interface and coordination which offers the benefit to use MDE techniques and concepts. However, our approach needs to be improved to provide automatic interfaces creation at model level as well as automatic generation of coordination's model conforming to the coordination model defined at language level.

REFERENCES

1. A. Klepee. Software language engineering, Addison-Wesley Professional, 2008.
   D. Strüber, S. Jurack, T. Schäfer, S. Schulz. and G. Taentzer. "Managing Model and Meta-Model Components with Export and Import Interfaces." In BigMDE@ STAF, pp. 31-36. 2016.
2. S. Jurack, G. Taentzer. "Towards composite model transformations using distributed graph transformation concepts," International Conference on Model Driven Engineering Languages and Systems, Springer, Berlin, Heidelberg 2009 Oct 4 (pp. 226-240).
3. D. Strüber, G. Taentzer, S. Jurack, S. and T. Schäfer. "Towards a distributed modeling process based on composite models "International Conference on Fundamental Approaches to Software Engineering, Springer, Berlin, Heidelberg. 2013, March (pp. 6-20).
4. S. Živković, D. Karagiannis. "Towards metamodelling-in-the-large: Interface-Based composition for modular metamodel development."" Enterprise, Business-Process and Information Systems Modeling. Springer, Cham, 2015
5. K. Chen, J. Sztipanovits and S. Neema. Toward a Semantic Anchoring Infrastructure for Domain-Specific Modeling Languages" EMSOFT'05 September 19–22, 2005, Jersey City, New Jersey, USA, 2005.
6. K. Chen, J. Sztipanovits and S. Neema. "Toward formalizing Domain-specific Modeling Languages." Vanderbilt University Institute for Software Integrated Systems, presented at the Object Management Group (OMG) OMG's First Annual Model-Integrated Computing Workshop, 2004.
7. T. Clark, P. Sammut and J. Willans. Applied metamodeling: a foundation for language driven development. Middlesex University Research Repository, 2008
8. T. Degueule, B. Combemale, and JM. Jézéquel. "On language interfaces." In Present and ulterior software engineering, pp. 65-75. Springer, Cham, 2017.
9. OMG: Model driven architecture MDA Guide rev.2.0, 2014-06-01.
10. L. Bettini. Implementing domain-specific languages with Xtext and Xtend. Packt Publishing Ltd. 2016.
11. E. Syriani, H. Vangheluwe, R. Mannadiar, C. Hansen , S. Van Mierlo, H. Ergin. AToMPM: "A Web-based Modeling Environment." Demos/Posters/StudentResearch@ MoDELS. 2013 Sep 29; 2013:21-5.
12. JP. Tolvanen, R. Pohjonen, S. Kelly. "Advanced tooling for domain-specific modeling: MetaEdit+." In Sprinkle, J., Gray, J., Rossi, M., Tolvanen, JP (eds.) The 7th OOPSLA Workshop on Domain-Specific Modeling, Finland 2007 Oct.

13. Z. Molnár, D. Balasubramanian, Á. Lédeczi. "An introduction to the generic modeling environment." In Proceedings of the TOOLS Europe 2007 Workshop on Model-Driven Development Tool Implementers Forum 2007 Jun.
14. V. Viyović, M. Maksimović, B. Perisić. Sirius: "A rapid development of DSM graphical editor." In IEEE 18th International Conference on Intelligent Engineering Systems INES 2014 2014 Jul 3 (pp. 233-238). IEEE.
15. D. Steinberg, F. Budinsky, E. Merks, M. Paternostro. EMF: eclipse modeling framework. Pearson Education; 2008 Dec 16.
16. M. Fowler. Language workbenches: The killer-app for domain specific languages, 2005.
17. E. Schindler, K. Schindler, F. Tomassetti, AM. Sutii. "Language workbench challenge" 2016: the JetBrains meta programming system. In LWC@ SLE 2016 Language Workbench Challenge, Splash2016, 39 October-4 November 2016, Amsterdam, The Netherlands 2016 Nov 2.
18. N.Essadi, A.Anwar, and Y.Laghouaouta, "Operators role-based approach to coordinate between heterogeneous DSLs". In Wireless Technologies, Embedded and Intelligent Systems (WITS), 2017 International Conference on(pp. 1-7). IEEE, April 2017.

AUTHORS PROFILE

Naima Essadi is a PhD student in SIWEB Lab belonging to Department of Computer Science and Engineering, Mohammadia school of engineers University Mohammed V in Rabat, Morocco. She received a Master degree in Computer Science and Telecommunication in 2007 from Faculty of Science University Mohammed V in Rabat. She worked for many years as software development Engineer at Alcatel-Lucent and at Moroccan Ministry of Justice. Her area of interest includes software Engineering, Model Driven Engineering as well as Domain Specific Modeling Languages.

Dr. Adil Anwar is an associate Professor in Computer Science at Mohammadia School of engineers, University Mohammed V in Rabat, Morocco. His area of interest includes software engineering, Model Driven engineering, Requirement engineering as well as Domain Specific Modeling Languages.