Architecture description language for Cyber Physical Systems analysis: a railway control system case study

N. Darragi, E. M. El-Koursi & S. Collart-Dutilleul
Université Lille Nord de France and IFSTTAR-ESTAS, France

Abstract

Cyber Physical Systems (CPSs) are the next computing revolution and the new generation of complex System of Systems (SoSs). CPSs are complex and ubiquitous embedded devices coupled with global integration respecting Moore's Law. Therefore, to fit with their new characteristics, we are facing several challenges, such as the proliferation and the integration of these systems into scalable environments.

A main concern of embedded real-time systems is safety. For such safety-critical systems, not only correct results count, but also the runtime duration for producing them. To ensure the dependability of such systems, which is not a local property of the system, but a global system property, the SoS safety has to be assessed, evaluated and checked according to its specific runtime context.

In an attempt to address the challenges, we propose two domain specific languages for modeling the system architecture and the dynamic behavior of heterogenous systems and their interactions. This paper shows how to develop an approach of real-time system design based on an extension of Milner's Calculus Communicating Systems since languages which are based on process algebra provide suitable features to formalise components communications.

1 Introduction

Cyber Physical Systems (CPSs) are physical engineered systems whose operations are monitored, coordinated, controlled and integrated by a computing and communication core. The need to specify the architecture model of such systems is justified by five main benefits [1]; the comprehension of the architectural

WIT Transactions on The Built Environment, Vol 135, © 2014 WIT Press
www.witpress.com, ISSN 1743-3509 (on-line)
doi:10.2495/CR140181

description at a level of abstraction, the reuse of the design pattern at multiple levels, the identification of the system components and their dependencies, the simplification of the evolution process and finally the provision of features to analysis and consistency checking.

Development paradigms, such as object- oriented, component- oriented and agent- oriented provide several mechanisms of abstraction, polymorphism and encapsulation to describe CPS architectures, which have an impact on the system architecture appearance within the whole complex implementation. The architecture description language (ADL) is used to define the borders and bindings between system components. According to [2], an ADL is a formal or semi-formal notations that could be textual or graphical allowing to specify system dynamic architecture and behavior. In the literature, several ADLs that rely on different paradigms, have been developed to model system architecture and system analysis and design. Some examples of well-known ADL are π-ADL [3] which is based on the higher order typed π-calculus, AADL for (Architecture Analysis & Design Language) [4], ACME/Plastik [5] which rely on first order predicate logic, Dynamic Wright [6] which is based on communicating sequential processes, etc. Several studies on ADL classification and comparison [7,8], demonstrate that these ADLs support one or many features as below and not all of them; the architecture description, the behavior modeling, the dynamic reconfiguration of the architecture (i.e., the modification of a software at runtime).

In this work, we propose an ADL which relies on Milner's π-calculus which is an extension of Calculus for Communicating System (CCS) [9] and which handles stochastic models. The ADL is based on a history-based specification approach so-called GORE for Goal Oriented Requirement Engineering [10]. The conceptualisation of the ADL with labelled transition systems is to capture a meta-model or core model of structural and dynamic system architecture. The benefits of GORE approach are various, such as the natural structuring of complex requirements specifications in different formal levels (Semi-formal and formal modeling). In fact, in GORE, goals are considered as perspective assertions that should hold in the system concerning software-to-be, execution environment, domain properties, expectations,... Goals provide also a precise criterion for sufficient completeness and pertinence.Furthermore, a goal model may easily express the complex constraints or the complex dependent behavior between agents of the system. IPL supports the dynamic reconfiguration of embedded and mobile systems.

In this paper, we give a brief description of our framework INSAC (INtelligent SAfety Checker) which use the proposed ADL. INSAC is based on GORE methodology which used to express system requirements in a high level of abstraction in terms of objectives and goals to achieve under constraints. It uses semantic nets for conceptual modeling of goals (services or quality of services), agents (active system components), objects (entities, relationships, events) and operations (Input-Output relations over objects), the first-order logic for the specification of goals and objects and state-base specifications for operations.

Our works focus on the verification and validation of CPSs. Therefore, we proposed the framework INSAC for the specification of CPSs requirements , the modeling and the formal verification. In this paper, we present a high-level architecture description language (ADL) to describe the complex architecture of CPSs and the scalable runtime environment interactions by using a description language IPL (INSAC Prescription Language), and second, to describe the dynamic of the system by using the modeling language IML (INSAC Modeling Language).

This work is structured as follows. Section 2 describes the framework, its scope and its architecture. Section 3 motivates this work thanks to the example of railway telecommunication system GSM-R of the European Railway Traffic Management System (ERTMS). The GSM-R architecture is illustrated by making use of the proposed ADL. In the same section, an extension of IPL so-called IML for INSAC Modeling Language is presented in order to describe the dynamic behavior model of the CPS.

2 Intelligent safety checking framework

2.1 The scope

The INSAC framework focus on the analysis, the design and the simulation processes of the system life-cycle. From the system requirement specifications (SRS), structured requirements with templates guided by a context-free grammar and based on a domain-specific ontology, are generated [11]. The analysis process is verified and validated by a testing method [12]. The second process is the design of specifications which provides an offline model of the SoS describing the architecture and the dynamic behavior of each system coupled with other systems. This step is also verified and validated by several approaches, such as well-known and widely-used provers and model-checkers in a comparison study. The last step is the dependability verification and the safety checking of the CPS based on the simulation. This process provides online models which are the refinement of offline ones. The simulation is based on MAS (Multi-Agent Systems).

2.2 The architecture

A component by definition [13] is “a piece of software offering (via an interface) a predefined service and which is able to communicate with other components”. A component may be composed of other components and encapsulated for a longer lifetime of implementations. Components allow a multiple-use which means the support of distributed and parallel execution of sub-services. It is non-context-specific. Therefore, it could be exchangeable and it allows the linkage avoidance of different software units.

In fact, a component is a trend to describe mechanical or electrical systems without the need to know details about all components of the system and the SoS. The model of a component that we proposed is shown in figure 2. The

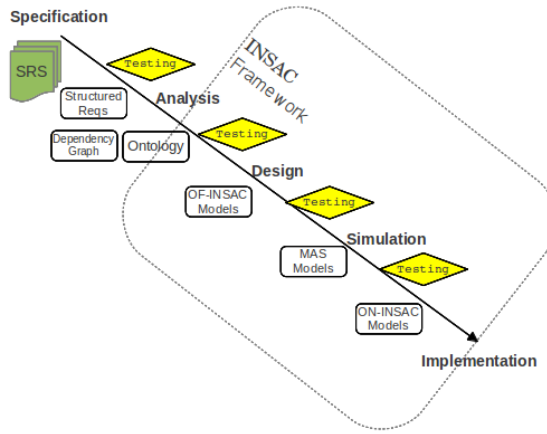

Figure 1: The scope of INSAC framework.

component model shows its main characteristics, such as the signature or the identification ID, the type and the category of the component named in the figure 2 role, tasks which represent functional requirements, states of the components or the functioning modes, used resources which could be local or distributed data or objects, constraints of the run-time execution which represent in reality the non-functional requirements, such as the safety requirements and, finally, what we call here by communication protocol, which is the interaction strategy with other components.

CPSs are complex SoSs composed of several components with hard time constraints. New SoS designs should take into account the complexity of coupled models of different involved systems and also unexpected interactions with components from run-time environments. The component model (non-bold shapes in figure 2) allows only the description of the human interactions with the system via interfaces, but not the behavior of the human agent as a "component" of the system which could be identified and defined as an entity that has an impact on the dependability and the safety of the system.

According to this criterion, a new definition of system "components" is proposed and shown by figure 2. Due to the complexity of SoS architectures and interactions, it is very important to show the interaction between systems, which could be resumed on five modes; the computation components, shared resources, controlled plant, human operators or the larger environment.

We distinguish two categories of agents; *atomic* agents and *compound* agents. The first category or the so-called *atomic* agents are those that have a simple structure without explicit parallel composition but could perform sequential or parallel tasks. The second category concerns the so-called *compound* agents which are composed of at least two atomic agents working simultaneously.

New added characteristics are shown in bold in figure 2. The system components are represented by agents which could be physical, logical or human entities. An agent, according to [14, 15], is adaptive (i.e., able to understand), rational (i.e., entities that do "right things" given what they know [16]), autonomous entity that is able to communicate with other agents and to react within its environment. An agent belongs to an environment in which there is a "problem" to which rational agents are a "solution".

Typically, each agent has a set of capabilities (i.e., the set of methods that an agent could perform), tasks which are the same for components, but they are re-defined to support the new agent characteristics (see figure 2), goals (i.e., the non-functional requirements concerning the quality of services that should be held until the execution of tasks), a set of responsibilities, the knowledge (i.e., the internal data and rules which are the results of "social experiences" within the execution environment with other agents. Knowledge is changeable and adaptable to new constraints), belief concerns the vision of the agent on its environment (i.e., the set of environment states), a set of faults which are incorporated in the definition of the system agent.

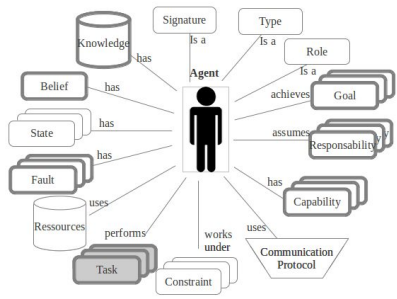

Figure 2: The INSAC agent model.

3 CPS architecture description using IPL

3.1 Introduction

The most important characteristic of distributed systems is communication. When there are communication between logical components or human and computational systems, there are some activities that should be taken into account, such as the establishment and the generation of the communication, the interpretation of responses and interactions, the understanding of these interactions and the internalisation of the knowledge issued and required for the communication by a learning process to be reused in the future with similar situations. Since the IPL is used to describe the logical, physical and human communication inside the SoS architecture, the proposed syntax plays an

important part in guiding the user in the complex architecture modeling process. IPL is a language with a grammar similar to the Extended Backus-Naur Form (EBNF) providing a specific language in order to avoid ambiguity, to have a brief and ordered descriptions.

3.2 Abstract IPL syntax

The IPL uses many concepts that are considered essential to define the dynamic architecture and, in future works, the dynamic behavior of CPSs. In ADL these concepts are presented as reserved words.

Agent represents a physical, logical or human entities of the SoS. It is an adaptive, rational entity that is able to communicate with other entities of the same "world" (i.e., System or Subsystem). Another definition says that an agent is an object with process characteristics and which is guided by its own goals and constraints.

Role is the main function or behavior of an agent which could be `processing`, `communicating`,...

Goal is the set of requirements that must be achieved, ceased, maintained or avoided during the execution of tasks. A goal is composed of a pattern [17], a status and a set of states used to determine a predicate (i.e., assertions on agent properties). This latter describes "Desires" of a BDI (Belief-Desire-Intention) agent [18]. A goal describes situations that are desirable for the agent.

States define the belief of a BDI agent which is a predicate describing a set of states.

Capabilities represent services, actions and abstract plans which the agent is able to perform.

Faults represent failure modes of an agent. A fault has a type, a priority{`Low, Medium, High, Critical`}, a status{`corrected, non-corrected, under-correction`}

Responsibilities indicate if the agent is mainly responsible for maintaining a goal or achieving a plan. This is used only in the case of a shared goal or plan.

Resources designates characteristics that an agent needs to perform a task.

Bindings are dynamic interactions (i.e., a single or sequence of events or actions exchanged between agents according to a specification. It designates the agent interfaces which include actuators (or effectors) and sensors. Called also connector, it is an architectural entity which defines connections between agents.

Knowledge is a structured base of concepts representing information and data acquired by the agent about its environment and formal rules (i.e., a set of condition-action rules) determining how these data represent objects or attributes of the current or other agents and their relationships.

CommunicationProtocol is a domain-specific ontology composed of structured terms, which describe words used to communicate between the sender and the receiver, and rules determining how terms are structured and used.

Configuration is a topology of agents describing how connections are established between them.

Plan defines a sequence of sequential or parallel tasks.

The relationship between agents are managed by some reserved words as below; "*isA*" designates the instantiation of an agent from a category of entities. "*as*" designates the nature of an agent which could be `HumanEntity`, `PhysicalEntity`, `LogicalEntity`. "*belongsTo*" is used to represent the hierarchy of the agent (i.e,. its instantiation).

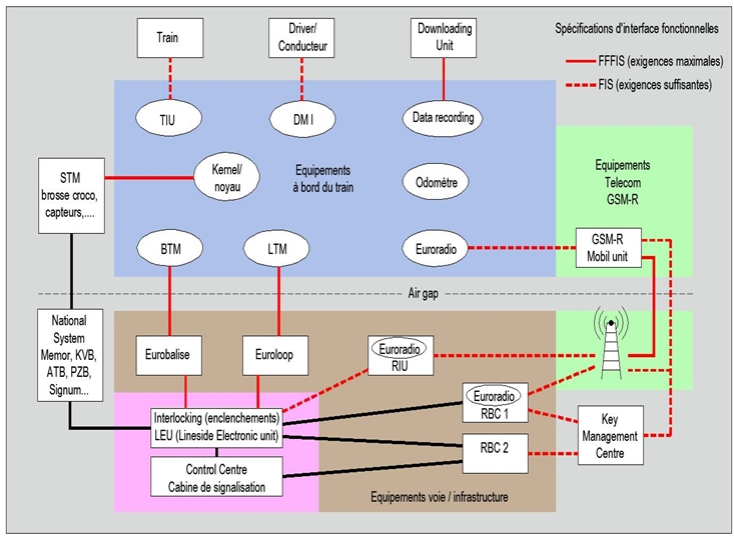

Figure 3: The GSM-R in ERTMS infrastructure.

3.3 GSM-R specification using IPL

The European Railway Traffic Management System (ERTMS) is composed of three major systems; the train-based computer ETCS (European Train Control System), the system of telecommunication GSM-R (Global System Mobile communication for Railway) used for communication on-board/Trackside and vice versa, and the traffic management system ETML (European Traffic Management Layer) which is currently still in the demonstration phase.

Listing 1: ERTMS Architecture using IPL

```
ERTMS isA Configuration;
Agent ETCS belongsTo ERTMS;
Agent ETCS–ONB belongsTo ETCS;
Agent ETCS–TRS belongsTo ETCS;
Agent GSM–R–System belongsTo ERTMS;
Agent GSM–R–FN belongsTo GSM–R–System
        belongsTo ETCS–TRS; /*Fixed Network*/
Agent GSM–R–MRD works GSM–R–System works ETCS–ONB:
/*Mobile Radio Device (with sim card)*/
        Role:= Communicating.
        Goal:= "Achieve a secured message transmission".
        States:= {waiting, receiving, sending,
                  establishingFail, indicationLosing}.
        Capabilities:= {startService(), finishService(),
                reqData(), alert(), reportPosition(),
                connect(), deconnect()}.
        Bindings:= {EffectorPort, SensorPort}.
        Faults:= {connectingError, receivingMSGError,
                  sendingMSGError, processingError,
                  connectionLosses, transmissionError,
                  handoverError, failedToActivate,
                  failedToConnext, failedToTerminate}.
        Responsibilities:= processingLocalMSG.
        Resources:= SimCard,
        Knowledge:= {CryptologyKey, HomeLocationRegister,
                IMSI, MSISDN, PIN,
                PIN2, RoamingData, MCC, MNC, TMSI}.
        CommunicationProtocol:= GSMROntology.
;
Agent ETCS–MERID works ETCS–ONB:
/*Mobile Euro–radio interface device*/
Agent ETCS–FERID works ETCS–TRS;
/*Fixed Euro–radio interface device*/
Agent RBC works ERTMS;
Agent BSS belongsTo GSM–R–System;
/*Base Station System Definition*/
Agent BTS belongsTo BSS:
        Role:= Communicating.
        Goal:="Achieve a secured message transmission".
        States:= {waiting, receiving, sending}.
        Capabilities:= {receive(), send(), startService(),
                        finishService(), estConnection(),
                        authentification(), TCHAssgnment(),
            callConfirmation(), setCipheringMode(),
            alert()}.
        Bindings:={SensorPort, EffectorPort}
        Faults:={connectingError, receivingMSGError,
                        sendingMSGError, processingError}.
        Responsabilities:= achievingTransmission.
        CommunicationProtocol:= GSMROntology.
;
```

To allow interoperability and to ensure the transition between different signalling systems, trains are equipped by embedding subsystems so-called EVC for European Vital Computer and the specific system of transmission, STM, for the Specific Transmission Module.

In ETCS level 2, the trackside is composed of many elements, as shown in figure 3, such as the Radio Block Center or Controller (RBC) which is the management module of radio transmissions between trackside and on-board for a limited area and the entity that delivers the movement authority (MA). RBC handles entry and exit of trains of the controlled area including the handover, train localisations, the allocation of free tracks. It handles perturbations and incidents, deals with real-time data and requests for MA and works as an interface for trains with different and various communication protocols.

3.4 IPL extension for modeling dynamic behavior

A CPS is a collection of heterogenous models and specifications. Each model is a formal description of a system of subsystems. CPS is a set of systems working concurrently with interactions, which is central to distributed real-time systems such as embedded. The extension of IPL will be the IML for INSAC Modeling Language and which is dedicated to describe the dynamic behavior of the CPS.

We distinguish two different models; the *atomic model* (AM) describes the dynamic behavior of every *atomic* agent and the so-called *cyber model* (CM) which is a coupled model driven by goals and operations where cyber capabilities (i.e. integration capability) are performed by agents. The CM describes the dynamic behavior of *compound* agents. The communication of every agent with its environment is via interfaces or communication ports called also *channels*.

4 Conclusion

Due to the size and the complexity of CPSs, many challenges are encountered when checking the safety of systems. First, we have a large amount of system information and knowledge to extract, to handle, to understand and to design in different levels of granularity and various perspectives.

This paper presents the INSAC Prescription Language which is a high-level domain specific language destinated to design cyber physical systems and the complex interactions between components called agents. The goal-oriented requirement engineering is used to provide the framework methodology of INSAC which use IPL at different levels of modeling.

IPL allows the description of the architecture of concurrent systems modeled by multi-agent systems. We are working on IML, which is an extension of IPL.It is supposed to provide features to describe the dynamic behavior of system agents and their interactions and communications. They will support various levels of abstractions (from high level specification to a fine granular level) which allow the description of the concurrency, the task distribution, the real-time environmental characteristics, the timed agents, stochastic tasks and, finally, synchronous and asynchronous communications.

References

[1] Fulkner, S., Kolp, M., Wautelet, Y., Achbany, Y.: A formal description Language for MA Architectures.In AOIS2006, LNAI 4898, pp.143-163, Springer-Verlag (2008)
[2] Garlan, D., Monoroe, R., Wile, D.: ACME: Architectural description interchange language. In Proceedings of conference of the Centre for Advanced Studies on Collaborative research. IBM Press (1997)
[3] Medvidovic, N., Taylor, R.:A classification and comparison framework for software architecture description languages. IEEE Transactions on Software Engineering, 26(1):70-79 (2000)
[4] Feiler, P., H., Gluch, D., P., Hudak, J., J.: The Architecture Analysis & Design Language (AADL): An Introduction. Publisher: Software Engineering Institute. CMU/SEI-2006-TN-011. (2006)
[5] Garlan, D., Monoroe, R., Wile, D.: ACME: Architectural description of component-based systems. In Foundations of Component-based Systems, pages 47-68. (2000)
[6] Allen, A., Douence, R., Garlan, D.: Specifying dynamism in software architectures. In proceedings of the workshop on Foundations of Components-Based Systems, pages 11-22. (1997)
[7] Kacem, M.H., Jmaiel, M., Kacem, A.H, Drira, K.: Evaluation and comparison of ADL based approaches for the description of dynamic of software architectures. In proceedings of the Seventh International Conference on Enterprise Information Systems, pages 189-195. (1997)
[8] Minora, L.,A., Buisson, J., Batista, T., V., Oquendo, F.: Issues of Architectural Description Languages for Handling Dynamic Reconfiguration. In Proceedings of CAL'12. (2012)
[9] Milner, R.: A Calculus for Communicating Systems, LNCS92. (1980)
[10] Lamsweerde, A., V.: Formal specification: a roadmap. ICSE - Future of SE Track: 147-159 (2000)
[11] Darragi, N., Collart-Dutilleul, S., El-Miloudi, E.: Requirements Specification Methodology Based On Knowledge Engineering: A case Study of Railway Control system. Journal of Information and Knowledge Management. ISSN (Paper) 2224-5758, pages 37-47 (2014)
[12] Darragi, N., Collart-Dutilleul., S., El-Miloudi, E.: Modeling and Verification Methodology for Control Systems. In proceeding of Transport Research Arena Europe. To appear (2014)
[13] Szyperski, C.: Component Software: Beyond Object-Oriented Programming. ISBN 0201178885
[14] Gasser, L., Rouquette, N., Hill, R., W., Lieb, J.: Representing and Using Organizational Knowledge in Distributed AI Systems. In Les Gasser and Huhns, M.N., Distributed Artificial Intelligence, Volume II. Pitman Publishers, Ltd., London (1989)
[15] Ferber J., Eco Problem Solving: how to solve a problem by interactions, In proceedings of 9th workshop on Distributed Artificial Intelligence. (1989)

[16] Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach. ISBN-13: 978-0-13-207148-2. Third Edition. Pearson Edition
[17] Darragi, N., Bon, P., Collart-Dutilleul, S., El-Koursi, E.: Tropos for Embedded Real-time Control System Modeling and Simulation. In Proceeding Workshop on Analysis Tolls and Methodologies for Embedded and Real-tie Systems. (2013)
[18] Bratman, M., E.: Intention, Plans, and Practical Reason. CSLI Publications. ISBN 1-57586-192-5. (1987)