Design Time Methodology for the Formal Verification of Intelligent Domotic Environments

Fulvio Corno and Muhammad Sanaullah

Abstract. Ambient Intelligence systems integrate domotic devices and advanced control and intelligent algorithms, thus leading to integrate systems with a high degree of complexity in their behavior. Ensuring the correctness of the design of such system is therefore essential, and this paper proposed a methodology, based on formal modeling and verification techniques, to verify logic and temporal properties of an intelligent ambient. The approach is integrated with the Dog domotic gateway, and automatic translation tools ensure the correctness of the verified model while adding no additional task for system designers.

Keywords: Ambient Intelligence, Model Checking, Formal Verification, Statecharts.

1 Introduction

Ambient Intelligence is a promising and quickly expanding research field and is expected in the near future to have a wide impact over people's lifestyles [8]. Major electronic companies and various research and development organizations are working on the production and development of these systems [1]. In this context, current technology (often called *domotics*) already enables today's homes to perform automatic and intelligent behaviors, by means of networked electrical devices (sensors, actuators, consumer devices and various other controllable devices) in the home environment, governed by suitable algorithms, which work intelligently by also considering the presence and actions of people. Such algorithms are often implemented in an embedded computer called *home gateway*. We call this currently available home intelligence solutions: Intelligent Domotic Environments (IDEs) [2].

Fulvio Corno · Muhammad Sanaullah
Politecnico di Torino, Torino, Italy
e-mail: {fulvio.corno,muhammad.sanaullah}@polito.it

Correct design of these IDEs is a challenge, since the interactions of many interconnected intelligent devices (composing the domotic systems) with the governing algorithms running in the home gateway may create complex global behaviors, that are difficult to predict before actually building and testing the system. System designers need powerful methodologies and tools to enable them to ensure the correctness of the system in early stages of the design process.

Some efforts to enable design-time validation propose the adoption of simulation based approaches, where a model of the system is expressed in an operational formalism that allows its simulation [4].

The effectiveness of simulation-based validation approaches is often not sufficient when critical systems are considered, where the domotic system must ensure some security -or safety- related properties, as is often the case when dealing with humans and their home environments. In this context, we advocate the use of Formal Verification techniques, that can give mathematical evidence that the desired system properties are satisfied by any possible system evolution.

This paper proposes a design time methodology for the Formal Verification of Intelligent Domotic Environments. The proposed approach is based on abstractly modeling the system as a set of concurrent UML 2.0 State Charts [5], that model the behavior of intelligent devices, the network connecting them, and the governing algorithms implemented in the home gateway. Such system is formally defined thanks to State Charts semantics [9], and can be formally verified by checking logical properties expressed in Temporal Logics [6, 7] by suitable model checking tools [11].

Formal Verification approaches are able to prove some properties of the provided models (in our case, state charts), but there is always the issue of ensuring the consistency of the verified model with the actual implemented system. This issue is solved in our approach thanks to the combination of:

1. ensuring the correctness of the state charts model of the devices and their communications using an automatic translation from the same system configuration description file that will be used at run time by the home gateway; this is based on the DogOnt [3] ontological description.
2. empowering the home gateway to interpret state charts at run time, so that the very same state charts for the governing algorithms that were used in formal verification will be used in the running system.

Verification techniques presented by J.C.Augusto at. el. [1] are based on Automata modeling of different devices IDE system, they also apply temporal properties for the verification of IDE system.

The remainder of this paper is organized as follows: basic description of the important components of the methodology is given in Section 2, and the main methodology is presented in Section 3 with a description of the adopted formalisms and tools. A case study showing the application of the methodology is presented in Section 4 and the associated verification results are discussed in Section 5. Concluding remarks and future work are finally discussed in Section 6.

2 Building Blocks

In this section, a brief introduction of DogOnt, Dog, State Charts and Temporal Logic is given, on which our verification methodology is based.

DogOnt [3] provides formal modeling and suitable reasoning for home environments through semantic web technologies. DogOnt is an ontological representation of the information of house layout and devices (which can be in real world IDEs) with their location, states and functionalities through semantic relations. The main focus is on the functionalities and states of the controllable devices in the IDEs.

Dog (Domotic OSGi Gateway) [2] is a domotic gateway for managing different domotic network components and their inter/intra communication and computational capabilities in a technology independent manner. Dog adopts the standard OSGi (Open Source Gateway initiative) [13] framework and it has capabilities for supporting the knowledge representation of semantic approaches and technologies; it can host intelligent applications.

State Charts are the Unified Modeling Language (UML) artifact, which is used for the graphical modeling of object-oriented softwares [5]. These are used for representing the dynamic aspect of the system, since the behavior of reactive systems can be modeled with the help of these State Charts.

DogSim [4] is an API for the automatic generation of State Charts. It takes the information of devices and their interconnection (event-messaging) from DogOnt, and with the help of Template Library Files (of devices) it translates this information into State Chart XML (SCXML) [14] files. These SCXML files are the state charts of devices and ready to be used for simulation or verification.

Temporal logic is a formalism widely used in Formal verification. It is a system of rules for reasoning with the different propositional quantifiers in terms of time. In the presented methodology, we use UCTL [12, 7], a UML-oriented branching-time temporal logic, which has the combined power of ACTL (Action Based Branching Time Logic) and CTL (State Based Branching time logic). UCTL uses the box operator ("necessarily," represented here as $[\![\,]\!]$) and the diamond one ("possible," $\ll\gg$) operator from Hennessy-Milner Logic and uses all temporal operators from CTL/ACTL (like Until, Next, Future, Globally, All, Exists).

3 Proposed Verification Methodology

As already stated, the goal of the proposed methodology is to verify functional properties of a given IDE, especially when it contains one or more control algorithms, which cause the overall system to exhibit complex behaviors. Our proposed approach, graphically summarized in Figure 1, is based on the following main assumptions:

1. the IDE is modeled according to the DogOnt ontology, that describes the Domotic plant composed of all the devices, their states, events and actions;
2. the control algorithms governing the IDE intelligence are expressed in the form of State Charts;

3. the residential gateway is able to process DogOnt system models and to interpret at runtime the State Charts describing the algorithms. In this paper, we adopt the Dog gateway [2] for this purpose;
4. system specification is given in the form of Temporal Logic properties and a suitable model checker is available for State Chart models.

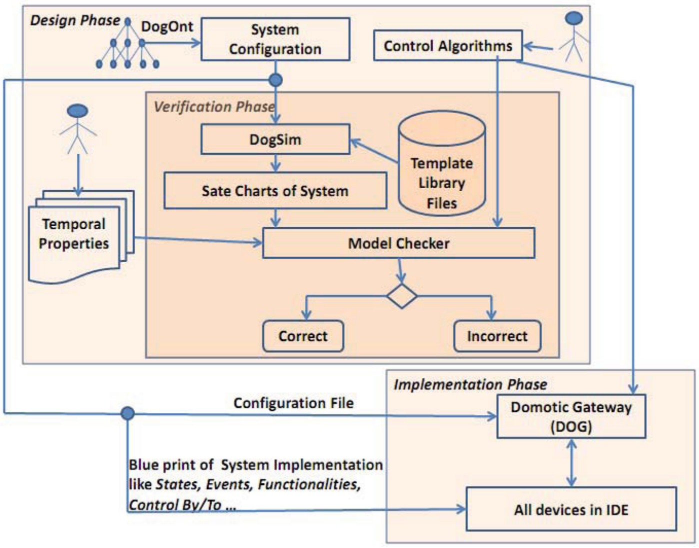

Fig. 1 Proposed Methodology

To apply model checking, we need a State Chart model of the domotic plant, corresponding to the DogOnt model. This can be obtained thanks to the DogSim compiler, that relies on a library of elementary State Chart templates describing each device's behavior. Such library templates are instantiated according to the devices listed in the DogOnt model and they are interconnected through 'connector' State Charts that model their communications and message exchanges; more details are available in [4].

Except for very trivial systems, the designer needs to implement one or more control algorithm(s) for embedding the necessary intelligence. We require such algorithms to be described in the form of State Charts. In general, deep analytical skills are required for the correct development of such control algorithm(s), and consequently for the correctness of the overall intelligence of the domotic environment.

The constraints and the behavior of the system, which is required to be checked, must be described in Temporal Logic. It is suggested [10, 11, 7, 12] that branching time Temporal Logic is best suited for the verification of State Charts, because it deals with multiple time lines. This manual property specification step requires a very skilled designer with the understanding of ACTL/CTL and Hennessy-Milner Logic. This step may require a significant amount of specification work, but is essential in the verification of any critical system.

Model checking can now be readily applied by verifying the temporal properties against the model consisting of the control algorithms coupled with the (automatically generated) State Charts of the domotic plant. The results of model checking will highlight the design errors, in the control algorithms or in the domotic plant, which should be located and corrected. The applicability of model checking to large systems needs to take into account the state explosion problem, and to address it with proper partitioning and abstraction techniques.

When all properties are successfully verified, we may safely move to the implementation step. The domotic plant, containing all the devices (that are assumed to be fault free), is connected with Dog, that allows the verified control algorithms to query and manage the actual devices. In fact, the control algorithms run by Dog are exactly the same State Charts used in the verification phase. Also, the same DogOnt model, that was used for generating the domotic plant model, is also used for Dog startup configuration. Therefore, the same information that was verified in the verification phase is used in the implementation phase, which guarantees that the system will work properly in all the verified scenarios.

4 Case Study

A simple but significant example of security and safety critical IDEs is a Bank Door Security Booth (BDSB). The BDSB system, represented in Figure 2, has two doors, for avoiding the harmful (direct) access of the user to the bank. One door is outside the bank and known as external door, whereas the other door is inside the bank and known as inner door. These doors have individual door-actuators for providing the force for opening and closing the doors. A user can access the door by pressing touch-sensors (TSs). These TSs are placed very near to doors and must be accessible on the each side of door.

The Door Lock Control (DLC) is the intelligent component of the BDSB system, it resides in the gateway (Dog in our case) and controls the evolution of the system intelligently. It checks the states of the devices and the events that devices generate; by considering the constraints, it decides what it has to do for obtaining correctness and security of the system. The DLC must ensure different constraints on the functionality of the overall system, some examples are: both doors can’t be opened at same time; the request from any TS will not be processed again until it is acknowledged back after completing the task; a specific door will remain open for a definite time (after opening and before closing), so that, the people can cross the door, etc.

Let us consider a simple scenario, a user wanting to enter the bank: he/she can press T1 (the Touch Sensor on the outer side of the external door) and the request goes to the DLC. The DLC considers the current configuration of the system and (if all constraints are satisfied) it commands the external door actuator to open the door. Smoothly the external door will be opened until the door sensor provides the information to the external door actuator that the door is completely open. Now, the user can cross the door and will reach in the isolated space. After a given time, the DLC again commands the external door actuator for closing the door, and the door will be closed smoothly. For entering the bank he/she has to press T3, and the same door opening and closing process will be performed on the internal door and the user will be inside the bank. By pressing T4, the user can start the sequence for exiting the bank.

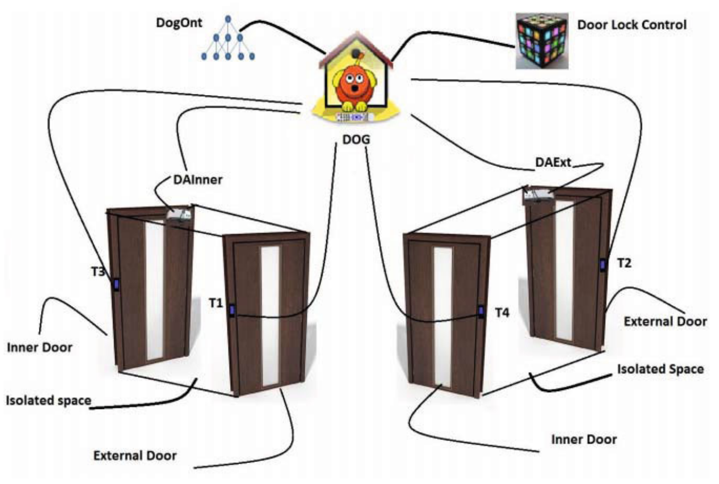

Fig. 2 Band Door Security Booth system with Dog

For the verification of these requirements, we select UMC (v 3.6) [10, 12] as a model checker because UMC supports state charts for describing the system, and for properties verification it can support UCTL.

5 Verification Results

After obtaining the State Charts of devices and control algorithm(s), it is required to translate these state charts into the input language of the chosen model checker: in our case, in the format of the UMC model checker. The designed model of the BDSB has 16,424 states. From the system specification, we derive different properties related to Security, Safety and Liveness on this model. For their verification, we evaluated nearly 50 different UCTL properties with the help of UMC framework. By considering the space limitation we here report just a few properties from the verified property set.

Property: *Against every posted request the specific TS must receive an acknowledgment*. The formula:

1. $AG[\![openRequest(T1)]\!]\ AF_{\{tsDone(T1)\}}\top$

is proven True by the model checker, which shows that if any sensor sends a door open request, sooner or later, it must receive an acknowledgment that the command has been executed.

Property: *TS will be available at anytime*. The formulas:

1. $AG[\![openRequest(T1)]\!]\top$
2. $AG[\![openRequest(T1)]\!]\ A\left[\top\ _{\{\neg openRequest(T1)\}}U_{\{tsDone(T1)\}}\ \top\right]$

are proven True by the model checker, which shows that TSs will be available at any time (*openRequest*), and a specific TS will not send another open request until it is acknowledged back after completing the task.

Property: *Interruption from any TS cannot break/change the execution of the current task*. The formulas:

1. $AG[\![openRequest(T1)]\!]\ AF[\![openRequest(T2)]\!]\ A\left[\top_{\{\neg daDoorOpen(DAExt)\}}U_{\{tsDone(T1)\}}\top\right]$
2. $AG[\![openRequest(T1)]\!]\ AF[\![openRequest(T3)]\!]\ A\left[\top_{\{\neg daDoorOpen(DAInner)}U_{\{tsDone(T1)\}}\top\right]$
3. $AG[\![openRequest(T1)]\!]\ AF[\![openRequest(T4)]\!]\ A\left[\top_{\{\neg daDoorOpen(DAInner)\}}U_{\{tsDone(T1)\}}\top\right]$

are proven True by the model checker. The set of three formulas satisfies the proper execution of T1, which means that when open-request of any door is in process and meanwhile another open request arrives from any other TS, the other request will not be executed until the first request completes its task.

Property: *Direct Access to the Bank is not possible*. The formulas:

1. $AG[\![daDoorOpen(DAExt)]\!]\ A\left[\top_{\{\neg daDoorOpen(DAInner)\}}U_{\{extDoorClosed()\}}\top\right]$
2. $AG[\![daDoorOpen(DAInner)]\!]\ A\left[\top_{\{\neg daDoorOpen(DAExt)\}}U_{\{innerDoorClosed()\}}\top\right]$

are proven True by the model checker, which shows that when one open-request is in process, if an open-request for the other door arrives, it can not be executed until the other door is closed, and therefore the doors cannot be open at the same time.

6 Conclusion and Future Work

Verification of the correct behavior of different heterogeneous devices integrated with control algorithms in Intelligent Domotic Environments improves safety, security and prevents critical threats. The presented methodology ensures the correct behavior of these IDEs with the use of Formal Model Checking techniques. In this paper, a small but not so simple case study is considered, and its correctness is proved against its temporal logic specification. The approach may also be applied to larger systems, even if scalability must be carefully ensured by partitioning and abstraction techniques usually adopted in model checking approaches.

In our future work, we plan to work on wider applicability and scalability of the approach, and on improving its automation. The problem of writing specifications will also be addressed, possibly by deriving temporal properties from ontology descriptions of the system.

Acknowledgements. This work is partially supported by the Higher Education Commission (HEC), Pakistan under UESTP-Italy/UET project. The authors thank Franco Mazzanti for his guidance and support.

References

1. Augusto, J.C., Mccullagh, P.: Ambient Intelligence: Concepts and Applications. Computer Science and Information Systems 4(1), 1–27 (2007)
2. Bonino, D., Castellina, E., Corno, F.: The DOG gateway: Enabling Ontology-based Intelligent Domotic Environments. IEEE Transactions on Consumer Electronics 54(4), 1656–1664 (2008), doi:10.1109/TCE.2008.4711217
3. Bonino, D., Corno, F.: DogOnt - ontology modeling for intelligent domotic environments. In: Sheth, A.P., Staab, S., Dean, M., Paolucci, M., Maynard, D., Finin, T., Thirunarayan, K. (eds.) ISWC 2008. LNCS, vol. 5318, pp. 790–803. Springer, Heidelberg (2008)
4. Bonino, D., Corno, F.: DogSim: A State Chart Simulator for Domotic Environments. In: 8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops), pp. 208–213 (2010), doi:10.1109/PERCOMW.2010.5470666
5. Booch, G., Rumbaugh, J., Jacobson, I.: Unified Modeling Language User Guide. The Addison Wesley, Reading (1998) ISBN 0-201-57168-4
6. Clarke, E.M., Emerson, E.A., Sistla, A.P.: Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications. ACM Transactions on Programming Languages and Systems 8(2), 244–263 (1986)
7. De Nicola, R.: Three Logics for Branching Bisimulation. Journal of the Association for Computing Machinery 42(2), 458–487 (1995)
8. Ducatel, K., Bogdanowicz, M., Scapolo, F., Leijten, J., Burgelman, J.C.: Scenarios for Ambient Intelligence in 2010. Tech. rep., ISTAG: IST Advisory Group (2001)
9. Gnesi, S., Latella, D., Massink, M.: Modular semantics for a UML statechart diagrams kernel and its extension to multicharts and branching time model-checking. Journal of Logic and Algebraic Programming 51(1), 43–75 (2002)
10. Gnesi, S., Mazzanti, F.: On the fly model checking UML State Machines. In: ACIS International Conference on Software Engineering Research, Management and Applications, pp. 331–3382 (2004)
11. Gnesi, S., Mazzanti, F.: A Model Checking Verification Environments for UML Statecharts. In: Proceedings of the XLIII Congresso Annuale AICA (2005)
12. Mazzanti, F.: UMC 3.3 User Guide, ISTI Technical Report 2006-TR-33. ISTI-NNR Pisa-Italy (2006)
13. OSGi Service Platform release 4. Tech. rep., The OSGi alliance (2007)
14. State chart XML (SCXML): State Machine Notation for Control Abstraction. Tech. rep., W3C (2010), `http://www.w3.org/TR/scxml/`