DoMAIns: Domain-based Modeling for Ambient Intelligence

(Article begins on next page)

04 August 2020

DoMAIns: Domain-based Modeling for Ambient Intelligence

Dario Bonino*,a, Fulvio Cornoa

a*Politecnico di Torino, Dipartimento di Automatica ed Informatica, Corso Duca degli Abruzzi 24, 10129 - Torino, Italy*

Abstract

Ambient Intelligence and Smart Home Automation systems are currently emerging as feasible and ready to exploit solutions to support more intelligent features inside future and current homes. Thanks to increased availability of off-the-shelf components and to relatively easy to implement solutions we are experiencing a steady evolution of households, causing an ever-increasing users' awareness of the capabilities of such innovative environments. To foster effective adoption of Smart Home Automation technologies in our home environments, traditional architectural and plant design must be complemented by sound design methodologies and tools, supporting the whole environment design cycle, including for example modeling, simulation and emulation, as well as, when feasible, formal model-checking and verification. Several research efforts have already addressed the design of expressive modeling tools, mostly based on Semantic Web technologies, as well as of suitable platforms for adding interoperation and rule-based intelligence to home environments. This paper proposes a new modeling methodology designed to fit the different phases of Intelligent Environments design, with a particular focus on validation and verification of the whole system. Carefully designed separation of modeled entities permits to exploit the DoMAIns framework during all phases of the environment design, from early abstract conception to the final in-field deployment. The DoMAIns design methodology is applied to a sample use case that involves comprehensive modeling and simulation of a Bank Security Booth, including the environment, the control algorithms, the automation devices and the user. Results show that the approach is feasible and that can easily handle different types of environment modeling, required in the different design phases, and for each of them it may support simulation, emulation, or other verification techniques.

Key words: Home automation, Intelligent Domotic Environment, Ambient Intelligence, State Diagrams, Domotic Plant, Modeling

1. Introduction

Ambient Intelligence (AmI) and Smart Home Automation (SHA) systems are currently gaining momentum by being recognized as affordable and easy to extend solutions for bringing intelligence to new and existing households. Although there exists a clear evidence of incompatibilities between domotic systems available on the market, researchers are converging on a common vision of SHA where traditional Home Automation (HA) plants built with off-the-shelf components are integrated by a single "computationally strong" element (device) supporting inter-plant interaction and bringing intelligence to the home [1]. Initial efforts [2, 3, 4, 5] involving these new Intelligent Domotic Environments (IDE) are nowadays maturing in a more structured research field including contributions from Ambient Intelligence [6], Pervasive Computing [7, 8] and Smart Environments [9, 10, 11]. Moving from first, sparse approaches, the research community is now tackling the design of next generation buildings and homes by applying well known, sound methodologies developed in the context of Software Engineering [12, 13, 14]. At the basis of this new wave of research lies the need to organically design models for complex Intelligent Environments [15, 16] and for the associated Context information [17, 18]. These models must permit, on one side, to address interoperability issues by exploiting a shared environment abstraction that enables the development of technology-independent home intelligence. On the other hand, thanks to formal representation of environments, devices, and behaviors, design models can be validated, verified and simulated prior to deployment, checking the compliance of envisioned solutions to se-

---

*Corresponding author

Email addresses: dario.bonino@polito.it (Dario Bonino), fulvio.corno@polito.it (Fulvio Corno)

curity, functionality, reliability and other requirements typical of such complex systems.

While context modeling originally attracted more attention, nowadays increased focus on home and device modeling has lead to several interesting approaches, mainly based on ontologies [19, 15] and web services. Only in the last years, modeling efforts started to focus on formal and dynamic models with the goal of supporting simulation and formal verification [20, 21, 22, 23]. However, additional contributions are still needed since most of available approaches limit the application of modeling and validation techniques to the early design stages and/or to specific, homogeneous subsystems. Conversely, the possibility of applying these tools throughout the entire design and development chain is seldom addressed, preventing the evolution of more comprehensive and integrated solutions, that could benefit form early verification as well as from development-time emulation, with some simulated devices and some real devices.

To overcome this issue and to support a more integrated and engineered design of Intelligent Domotic Environments, we propose a general and modular modeling approach that seamlessly supports different design phases and their corresponding configurations, by adopting modeling techniques that enable validation and verification at different stages. This design framework, called DoMAIns (recursive acronym for "Domain-based Modeling for Ambient Intelligence"), supports the whole IDE design chain, from early specifications to final on-the-field deployment. To support this ambitious goal, we strictly apply well known partitioning and modularity principles, striving to achieve a clear separation between different modeling aspects and concerns. Such a differentiation is crucial to support verification, simulation and emulation in the same model. By keeping modeling concerns (i.e., representation domains) separated, we expect the DoMAIns framework to successfully tackle complex, intelligent environments, supporting, in each case, clear identification of solutions under test and concurrent design and development of multiple, alternative ideas.

This paper contributes a first high-level view over DoMAIns modeling, basing on 5 pre-defined, yet extendable, *Representation Domains* and shows a concrete application example with the aim of offering a better understanding of the proposed methodology as well as to foster further investigations. Representation Domains are different, and possibly overlapping, sub-models of an Intelligent Domotic Environment connected through explicit "boundary" definitions or *interfaces*. Within each Domain, a given model is easily replaceable by

alternative ones (based on the same or on different formalisms, e.g. a statechart model can be replaced by a finite elements model, etc.), or by its "real" counterpart, i.e., by actual software modules, hardware devices or users.

Whenever real and simulated items co-exist in the same DoMAIns model, we refer to it as to an *Emulation Model*, i.e., a model in which elements under test (verification / validation) are connected to real items in an hardware-in-the-loop chain. Emulation models clearly require suitable adapters to convert information across the "virtual" and "real" domain elements.

Separating dynamic IDE models into Domains brings several advantages:

1. It permits to detail domain models with *different granularities*, depending on the representation goals, e.g., pure simulation, preliminary design verification, fine-tuning of control algorithms, etc.

2. It supports different *representation techniques*, enabling designers to tackle every domain with the most suited solutions, e.g., finite elements simulations for heat transfer, state machines for heating actuators, robust control theory for fine temperature regulation.

3. It provides clean *interface points* for cutting "virtual" IDE representations and for integrating real devices in the end-to-end simulation chain (emulation).

Clearly, many efforts are still needed to fully realize the vision underlying the DoMAIns approach, however this paper tries to lay down the general structure of Representation Domains on top of which more effective and comprehensive IDE modeling can be devised.

The remainder of the paper is organized as follows: Section 2 reports an overview of relevant related and complementary works. Section 3 introduces the representation domains defined in DoMAIns, describing their peculiar features and the interfaces between them. Section 4 shows use cases and applications in which DoMAIns can be successfully exploited, highlighting the flexibility of the proposed modeling methodology. Section 5 provides a modeling example where the DoMAIns framework is applied to the representation and simulation of a Bank Security Booth control, whereas Section 6 provides the relative implementation details. Finally Section 7 concludes the paper and proposes future works.
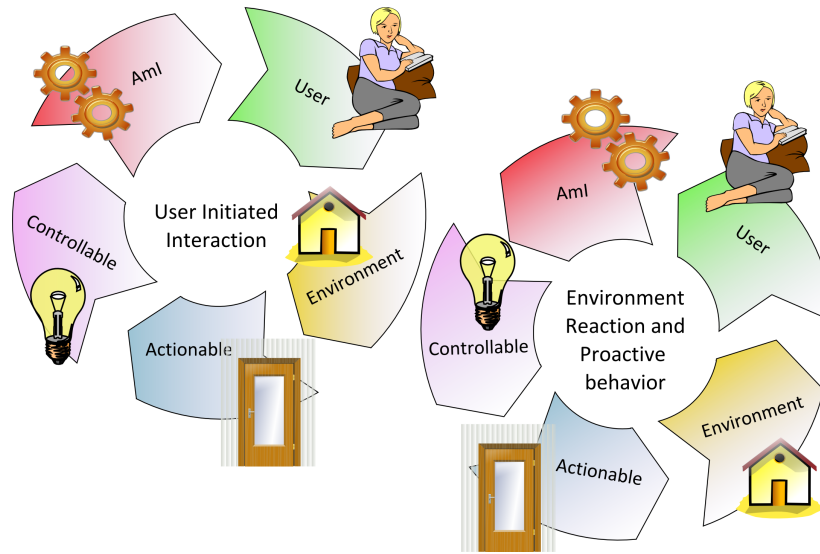
The Intelligent Building Ecosystem

Figure 1: The Building Ecosystem in terms of representation domains.

2. Related Work

Research in modeling, simulation and verification of Intelligent Environments is rather new and still consists of sparse and mainly un-coordinated efforts. Currently, few works try to tackle the whole IDE design life cycle, from early design stages to final in-field deployment. On the converse, several interesting approaches can be found dealing with specific issues relevant to single design/development stages.

Habitation [14] is currently, to our knowledge, the most relevant modeling effort tackling the whole IDE design process. It is a Domain Specific Language (DSL) explicitly designed for Home Automation and one of the earliest attempts to apply Model Driven Engineering to the Home Automation domain. It combines a model driven approach with DSLs to support the design of home automation systems, from high level, technology independent graphical design to technology-specific automatic code generation. The Habitation language is based on a three-layered approach including: (*a*) a computation-independent model, which represents the syntax and part of the semantics of the defined DSL, (*b*) a platform-independent model, which is a simplification of the UML meta-model for reactive systems, and considers components, activities and state diagrams, and (*c*) a platform-specific model in which a meta-model for KNX/EIB home automation technology is defined, exploiting the domain object model used by

the ETS[1] tool. DoMAIns differs from Habitation in both philosophy and technical details. On the philosophical point of view, while Habitation takes a clear vertical modeling approach, whose main modeling choices (and primitives) tend explicitly to semi-automatic code generation, DoMAIns adopts an horizontal framework approach aiming at defining easy separable domains involved in IDE design, leaving technological details to specific "instantiations" of the framework, depending on the specific definition of domain contents. On the technical side, Habitation does not cover issues related to validation, verification and emulation, by being strongly biased towards generating home automation systems. DoMAIns, on the converse, defines a clear set of representation domains that natively support design modularity and emulation scenarios. In addition it is purposely designed for supporting formal verification whenever all the domains required in a given modeling task are represented by compatible formal models, e.g., state charts. Finally, while Habitation is mainly targeted at programmable home automation systems, DoMAIns strives to be agnostic on the kind of "smart environment" thus being more flexible and supporting smart domotic systems made by not-programmable devices whose predefined behaviors have to be combined in new ways, or enhanced by an intelligent gateway controlling

---

[1]http://www.knx.org/knx-tools/ets/description/

them, for achieving more intelligent functionalities.

The V-PlaceSims [24] approach is another relevant example of "end-to-end" modeling approach for smart environments, specifically targeted to simulation. In V-PlaceSims, smart environments are modeled and simulated through Virtual Reality giving the users the opportunity to "directly experience" designed policies and smart automation solutions. By means of context-aware building data models, human-space interaction, which is vital for simulating smart home functions and services, is realized in a virtual environment. In addition, V-PlaceSims allows to materialize invisible services, performing real-time interactions with the home and making users aware of configuration possibilities, with respect to their real needs. Similarly to the Habitation case, DoMAIns adopts a clearly different approach to smart environment modeling: while V-PlaceSims tackles the whole design process through the lenses of user-home interaction, DoMAIns aims at supporting many different representation needs, also including user issues. However, rather than being in contrast with the latter, DoMAIns can be exploited to rationalize and made explicit the assumptions and needs lying at the basis on the V-PlaceSims approach, thus contributing to a cleaner and more integrated approach to smart home automation design.

DoMAIns applies to IDE design a modeling concern separation which is quite similar to Aspect Oriented modeling[2] [25, 26], in particular it can be considered as a specialized kind of symmetric AO[3] modeling for the smart home automation domain. Differently from Aspect Oriented Software Development, concerns are not relative to functionalities or software artifacts, but deal with real and desired behaviors and configurations of tangible environments. As a consequence, model weaving cannot be carried in a completely automated way and the contribution of a human modeler is required to tailor the framework to specific representation issues.

Several modeling techniques can be integrated in the DoMAIns framework, each related to specific modeling aspects and or design phases. Among the most relevant efforts, we can distinguish approaches related to context modeling (Environment and User domains) [17, 18], to user behavior learning and profiling (User domain), to usability evaluation [27], to the generation of "task-supportive" environments [28], etc. (see [6] for a quite complete review). Rather than being in contrast with DoMAIns, they are clearly complementary,

contributing to provide best suited modeling primitives within each domain defined in DoMAIns, depending on the target application and deployment.

3. Representation Domains

The domain-based modeling approach is quite general and may be extended and customized to different application requirements. In this paper we aim at showing one particular set of domains, suitable for the Intelligent Domotic Environments described in the introduction, where commercial off-the-shelf components are used to build an home automation system, which is enriched by intelligent computational elements.

In the IDE context, DoMAIns predefines 5 different representation domains (see Figure 1): User, Environment, Actionable, Controllable and AmI (i.e., Ambient Intelligence), typically sufficient to represent Smart Home Automation environments. These 5 domains emerge from the author's extensive experience on IDE design and from typical home configurations found in the literature.

Domains are neither exhaustive nor complete and they can be easily integrated by additional domains and/or refinements depending on the modeling tasks they are applied to. Nevertheless they achieve a degree of modularization sufficient to support most design tasks for Intelligent Environments as detailed in Section 4.

Representation domains play different roles in the intelligent building ecosystem by supporting a kind of closed loop interaction between users and intelligent environments (see Figure 1).

Domain definitions, given in the following subsections, identify the sufficient conditions for which a given environment entity belongs to one domain. We explicitly refer to *sufficient conditions* as different design needs might lead to different domain boundaries, while the proposed high-level subdivision is still valid. In each domain, either pre-defined or specifically designed, different representation techniques and formalisms are possible. To help the reader in understanding the abstract definitions of the Domains, in Table 1 we provide some examples of modeling techniques that are usually adopted in each of the 5 proposed domains. In every domain, the table groups some possible models according to their nature: real, ontology-based, simulatable, or formally tractable.

3.1. User

The user domain encompasses human inhabitants and their interactions with surrounding environment enti-

---

[2]AOSD website, http://www.aosd.net
[3]similar to approaches in CaesarJ (http://www.caesarj.org), and in CME (http://www.research.ibm.com/cme)
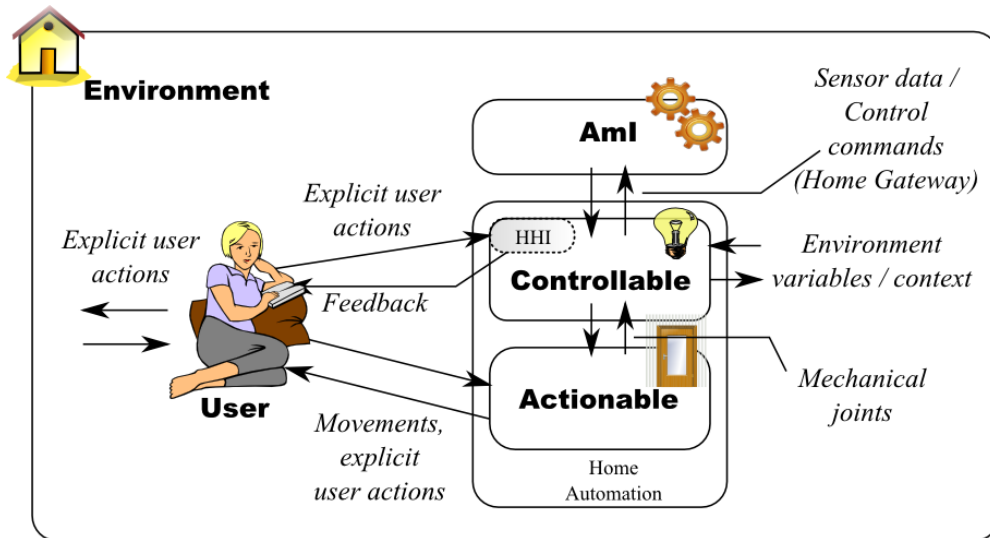
Figure 2: The 5 pre-defined domains and their relations.

ties, i.e., it refers to user behavior representation. User domain boundaries are defined by tangible interfaces between humans and the surrounding building environment (see Figure 2), be they:

- physical objects, e.g., doors, handles, buttons (Controllable/Actionable);

- physical quantities, e.g., heat transfer, acoustic or gas emissions (Environment);

- software artifacts, e.g., graphical user interfaces, mobile devices, pervasive and disappearing computers (Controllable).

Users can either be real or they can be described in terms of expected behaviors, by modeling typical interaction patterns involving the home environment. For example a suitable user representation may model the sequence of door openings and device activations that takes place when someone in the house is accomplishing a given task, e.g., cooking dinner. Users are central players of the intelligent buildings eco-system, acting both as source of requirements and as final consumers of local AmI policies. This two-folded nature is reflected by their role in the domain-based modeling approach we propose. In particular, the User Domain explicitly represents users as living in the modeled environment and acting on the environment entities (with their observable behaviors) to accomplish their goals, and this is clearly reflected by the user domain boundaries. We also remark that users are the ultimate targets of intelligent policies being designed, however this aspect is not explicitly modeled in the User Domain, but it rather drives the design process of AmI solutions.

3.2. Environment

The Environment domain includes all the building elements that have a *passive* or *background* role in the user-home interaction. The term "passive role" is referred to objects that are not directly actionable by electrical devices but whose properties can be changed or influenced by means of electrically controlled devices, in an indirect relation. For example, a room belongs to the environment domain since it is not directly actionable by some electrical devices, however its properties might be indirectly influenced by a home automation system, e.g., through a heater that may cause changes to the room temperature.

The Environment Domain encompasses two main object categories: fixed *architectural elements* such as rooms, walls, ceilings and floors and *furniture* like tables, chairs, and closets. The former domain subset acts as a "container" for the latter as well as for other, distinct, representation domains including User, Actionable and Controllable. While many fixed architectural elements assume a background role in user-home interactions, furniture is typically passive, unless enriched by "pervasive" interface elements (but in this case it would be considered as belonging to the Actionable domain). The Environment domain, depending on different representation needs, can be declaratively represented, fully simulated, real, or emulated. Figure 2 clarifies the interface and containment relations occurring between the

Table 1: Examples of representation techniques applicable in DoMAIns.

Domain	Real	Ontology	Simulatable	Formally Verifiable	…
User		foaf	Statecharts, Stocharts, HMM, Decision trees, …	Statecharts, Ontology, …	…
Environment		DogOnt, DomoML, Soupa	Statecharts, Finite Elements, Simulink,…	Statecharts, …	…
Actionable		DogOnt, DomoML	Statecharts, Software Simulators, …	Statecharts, …	…
Controllable		DogOnt, DomoML	Statecharts, Software Simulators, …	Statecharts, …	…
AmI	Software	DogOnt, Rules (SWRL, Jena), Inference	Rules(Drools), Matlab, Agents, Statecharts, …	Statecharts, …	…

Environment, the Actionable and the Controllable domain, highlighting (see smaller text on the diagram) that the last 2 can be seen, at an higher abstraction level, as belonging to a more general domain called Home Automation.

3.3. Actionable

The Actionable Domain includes environment objects that are (or that can potentially be) actioned by means of one or more electrically controlled actuators, e.g., doors, windows and gates. The main difference between Actionable objects and other Environment entities can be identified in the role that such elements assume in user-home interactions and in the way these elements are interfaced with the Controllable domain. First, Actionable objects are directly involved in user-home interactions, i.e., *the user directly acts* on these objects, e.g., opens a door or moves up a shutter. Secondly, the interface between objects belonging to this domain and the devices being part of the Controllable domain is typically composed of *mechanical bindings*, e.g., screws or rigid joints.

3.4. Controllable

The Controllable domain groups all the *electrical devices* that are either *part of the home automation/in-telligence hardware* (e.g., domotic plants) or that are integrated through a myriad of protocols and communication means. The distinguishing feature of a controllable device is its ability (either native or externally added, e.g., by suitable interfacing modules) to interface (or to be interfaced by) the main home automation and intelligence plant, i.e., the home gateway. Controllable devices share the common characteristics of being electrical, of having some kind of communication capability and of being able to "operate" on the environment for performing actions (actuators), measuring physical quantities (sensors), or to support user-home interaction; we refer to this last class of devices as to Human Home Interfaces (HHI) (see Figure 2). The domain boundaries are rather clear: on the Environment side they include physical bindings such as direct mechanical connections (e.g., screws attaching actuators to doors) or physical interactions (e.g. metal/air thermal exchange between a heater and the air contained in a room). On the AmI side, the domain boundaries are implemented by the Home Gateway, which interfaces and abstracts all the subsystems involved in a smart environment, offering a uniform access layer exploited by technology independent intelligence policies and algorithms.

3.5. AmI

The Ambient Intelligence (AmI) domain groups all the software solutions, i.e., algorithms, models, interfaces, that enable a given automated environment to become sensitive, adaptive and responsive to people activities and to events occurring in the home environment. The domain extension is almost indefinite, but the boundaries are well defined and correspond to the home gateway on one side and to users on the other. Building on top of the Controllable domain abstraction provided by the gateway, the AmI domain population is composed of modules, representation techniques, reasoning systems sharing the common goal of making the building environment able to *proactively, but sensibly, support people in their daily lives* [6]. We do not define specific approaches belonging to this domain, instead several contributions can be cited coming from many complimentary research fields involving sensing technologies, reasoning on environment models and/or actions, autonomous decision planning, spatial and temporal inference, human computer interaction and context awareness, security and privacy (see [6] for an overview of recent research efforts).

4. Domains in Action

Dividing the intelligent building ecosystem into different domains permits to effectively tackle different design activities, and associated issues, starting from initial, abstract design (fully simulated, for example) and reaching final deployment in the real world. By combining different modeling methodologies and detail levels in each of the various Domains, several activities in the design flow can be supported. Some, not exhaustive, examples are reported in Table 2, where we list possible use cases of DoMAIns models, by showing which Domains are involved, and what kind of model is required in each Domain. For many use cases, we also give references to published results.

Among the cited applications, some use cases have already been tackled by the authors in their previous works. For example, in [29] structural verification of IDE properties is addressed by exploiting the DogOnt ontology model for describing entities belonging to the Environment, Actionable and Controllable representation domains. On the other hand, the DogSim [23] simulator exploits the DogOnt ontology and a library of state-machine templates to offer IDE simulation capabilities, with a particular focus on design of automation plants. Finally, the EmuDog [32] framework extends

DogSim to support mixed interaction of real and simulated entities, i.e., it supports the emulation use case reported in Table 2.

Many combinations of simulated, real and formally represented domain entities are allowed. Whenever a given application scenario only involves ontology-based representations and/or other formal models carrying a well defined and verifiable semantics, simulation-based validation and formal verification can be performed [31, 32]. For example, state diagrams can be exploited to describe formally verifiable end-to-end automation scenarios, from user models to AmI. However, formal verification of a whole system is not guaranteed even in this case, due to model complexity that may grow beyond the checkers capabilities. In these cases, the availability of different models with different granularity, and the natural partitioning of the model in domains, may help taming the complexity.

The following section describes a sample model based on the DoMAIns framework that exploits state diagrams to provide an easy to simulate and verify representation of a bank security booth. While a complete evaluation of the approach would require modeling and implementing a much larger system, our case study is small and simple enough to be reported in the paper with high level of detail. The goal of the case study is to show how the methodology may be used, the types of involved models, and the kinds of design steps that are enabled by DoMAIns modeling.

5. The Bank Security Booth Example

We consider a very simple case study, namely a bank security booth automation to show how a real world system can be represented and simulated by applying the proposed representation framework. Thanks to DoMAIns, several control algorithms, sensors and actuators can be modeled and checked, in different configurations, supporting a complete design process which allows to have guarantees on the final result, deployed in the real world. For the sake of simplicity our sample model, while realistic, concentrates on the main function of the device and neglects some ancillary aspects such as accessibility, user flow rates, etc.

Bank security booths must respect a well defined set of requirements concerning protection from harm, access for people with disabilities, reasonable in / out flow rate and direct entrance / exit prevention (to prevent robbery). In the sample, we only concentrate on the last issue, i.e., on the control algorithms that must prevent users to be able to directly access or exit the bank

Table 2: Examples of domain-based modeling applications.

User	Environment	Actionable	Controllable	AmI	Use Case
-	-	-	S	R	Development of Automation Scenarios
-	S	S	S	R	Development of Control Algorithms [23]
S	S	S	S	-	(Automation) Plant Design
R	S	S	S	-	User testing (no AmI)
R	S	S	S	R	Human Home Interaction testing
-	O^F	O^F	O^F	-	Structural Property Check [29]
-	R+S	O^F	O^F	-	Smoke/Fire propagation estimation
-	-	-	$(O+S)^F$	-	Simulation Library Verification [30]
S^F	S^F	S^F	S^F	S^F	End-to-end AmI verification [31]
R+S	R+S	R+S	R+S	-	Emulation [32]
…	…	…	…	…	…

Legend: S = simulatable, R = real, O = ontology-based, ^{F} = formally verifiable

premises. These requirements can be synthesized as follows:

1. The bank security booth is composed of two doors with an "isolated space" between the external the internal entrance (Figure 3);

2. The 2 doors must never be opened at the same time (no direct access);

3. Doors must be "controllable" from inside the "isolated space," allowing users to decide to either enter or exit, and to prevent users from getting stuck in-between the 2 doors.

4. Time for entering / exiting the bank must be finite and higher than a minimum safety threshold.

A suitable bank security booth control algorithm must respect the above requirements, independently from user or door behaviors. Clearly we cannot rely solely on field testing since failing algorithms may either allow users to bypass security measures or they may cause harm to users, for example by locking them in the isolated space. Therefore, validation of such algorithms must be first performed in a fully simulated environment. If the environment is based on models carrying a well defined semantics, formal verification could also be applied by transforming some of the above requirements in specific constraints to be checked, e.g., as temporal logic assertions [31].

We provide here a state-diagram model that applies the DoMAIns framework to cover end-to-end the Bank Security Booth simulation: from User to AmI. State diagrams are chosen since they support both simulation and model checking [33], thus enabling designers to enforce and validate well-defined, strict requirements on the bank security booth control system. We suppose that the doors of the bank security booth are completely automated and do not allow direct operation, e.g., opening by turning an handle. Instead, they are attached to suitable actuators, managed by the Bank Security Booth control, which in turns interacts with the user by exploiting touch sensors (T1 ... T4) installed on the door frames.

5.1. User

In the Bank Security Booth scenario, a user can perform a variable set of actions. For example, upon entering she can:

a) decide to enter through the bank security booth or not;

b) if the door does not open in a reasonable time, she might decide to give-up;

c) once inside the isolated space, she can either decide to enter the bank interior or to exit.

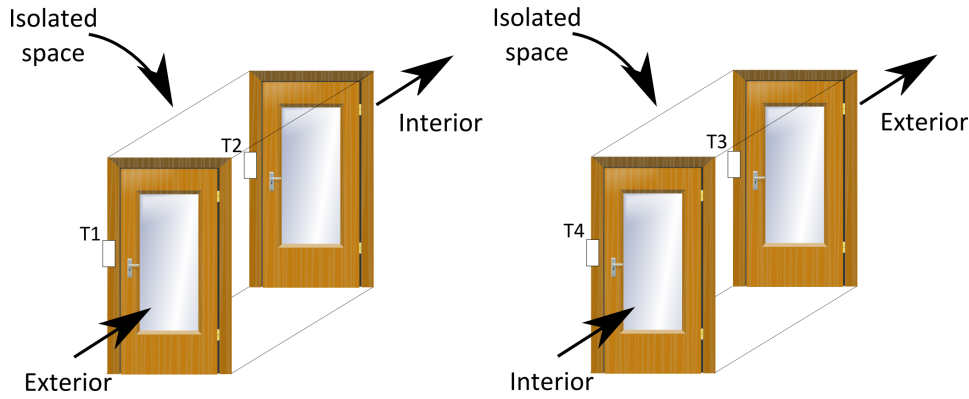A symmetrical set of actions can happen when exiting from the bank.

Figure 3: A typical Bank Security Booth layout.

To model this complex behavior through state diagrams, the plain UML2.0 specification is not sufficient and non-deterministic transitions must be addressed through a more expressive state diagram formalism called P-State Charts [34]. P-State Charts extend state diagrams by adding probabilistic elements: a transition is allowed to lead to one of several states depending on a probability distribution. The P-State Charts semantics is formally defined as a mapping on (strictly) alternating probabilistic transition systems [35], a subset of Markov decision processes (MDP) [36]. To allow verification of probabilistic temporal properties over probabilistic state diagrams, properties may be expressed in the probabilistic branching-time temporal logic PCTL [37], the prime logic for property specification and verification of models that exhibit both probabilities and nondeterminism.

Modeling the previously cited user behaviors by means of P-State Charts leads to the sample model reported in Figure 4. We adopt it as User domain model for the Bank Security Booth example.

As can easily be noticed, the user behavior is not deterministic due to the many *p* nodes in the diagram, where transitions to be executed are probabilistically chosen between two or more alternatives, with a distribution reported on the outgoing edges. The user model is characterized by a set of 4 outgoing events:

1. *entryRequest*, representing the user touching the external door handle or opening button (T1 in Figure 3);

2. *entryRequest2*, representing the user touching the internal door handle or opening button, on the isolated space side (T3);

3. *exitRequest* that models the user touching the internal door handle or opening button, on the bank interior side (T4);
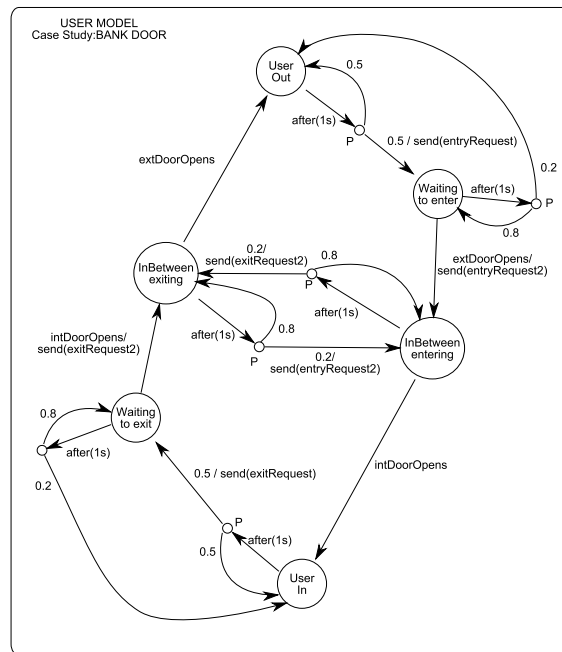

Figure 4: The P-State Chart User Domain model.

4. *exitRequest2*, which represents the user touching the external door handle or opening button, on the isolated space side (T2).

These events are exploited in the domain interface to connect the user model with models belonging to adjacent domains, e.g., the Actionable and the Controllable domains.

5.2. Environment

The environment domain exploits the DogOnt [38] structural part (`dogont:BuildingEnvironment`) to formally describe the booth architectural elements, i.e., the walls, the floor and the ceiling, the inner and outer spaces. Environment modeling through DogOnt allows performing static structural checks on the bank security booth design. For example, it allows to check whether the booth doors have all required touch sensors or not (see [29] for insights). In the Bank Security Booth use case, models belonging to the Environment domain do not take part in dynamic simulations shown in this paper. However, they play a relevant role in design validation allowing to verify constraints on the booth "architectural" configuration, in addition to the functional constraints defined for the booth control algorithm.

5.3. Actionable

Only two actionable objects are involved in the Bank Security Booth example, i.e., the exterior and interior doors. Their behavior can be easily modeled with two identical deterministic state charts. Figure 5 reports the corresponding diagram, where the three incoming events *dOpen*, *dClose* and *stop* represent the access point for the domain, i.e., the domain interface. Timed transitions between the 'moving' state and the 'rest' state simulate the time needed for a door to become completely open or closed.
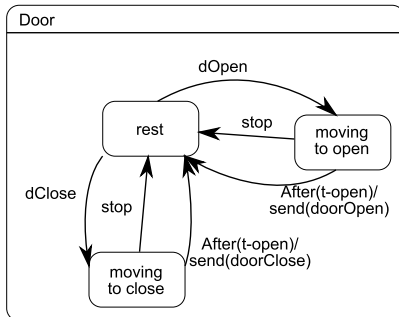
Figure 5: Sample state diagram of a door.

This simple model can easily be extended, for example by exploiting stochastic transitions, to represent possible door failures.

5.4. Controllable

In the Bank Security Booth example, we decided to show two different models for the Controllable domain. The two models presented here involve:

1. the first one, fully simulated entities (*simulation*);

2. the second one, partial integration of real devices (*emulation*) together with some simulated ones.

Involved objects include touch sensors, door actuators, and door sensors. The former two categories are both simulated (*simulation*) and real (*emulation*), whereas the latter is simulated in both variants.

Touch sensors are applied to each side of the booth doors and on the door handles (see Figure 3) and they support the control algorithm in detecting the user presence and desired actions, e.g., to open a door. They are extremely simple to model as their only "output" is a "touch" event used as a trigger by the door control algorithm. In the fully simulated variant touch sensors can therefore be easily represented by means of single-state machines, with only one self-transition (Figure 6). Instead, in the "emulation" variant, they are just substituted by domotic components behaving as normally-open contacts.
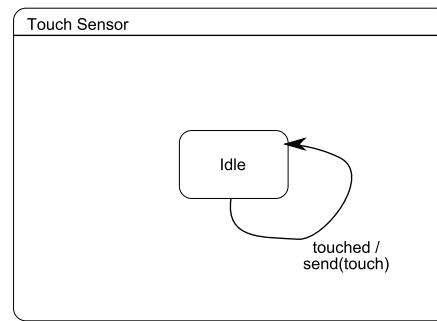
Figure 6: A touch sensor model (Controllable domain) in the Bank Security Booth scenario.

Door actuators are more complex, and their actual behavior depends on the implementation provided by each door actuator manufacturer. In this specific example we decided to model a prototype door actuator logic that derives from the actual door actuators commercialized by BTicino,[4] a leading Italian electric man-

---

[4] http://www.myhome-bticino.it

ufacturer. The resulting state diagram is reported in Figure 7. In the *emulation* variant, the actuator state chart is replaced by the real BTicino door actuator, installed in a demo case hosted in our Lab (Figure 13).
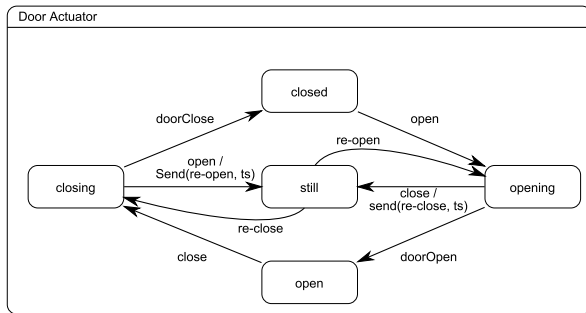

Figure 7: A door actuator model (Controllables domain).

5.5. Interfaces

To successfully accomplish end to end modeling in the DoMAIns framework, glue-layers between different domains shall be defined allowing to replace every single domain (or sub-domain) with its real counterpart (emulation support) or with alternative modeling techniques. In the DoMAIns terminology these layers are called interfaces; they can be automatically generated given a formal representation of involved domains (e.g., an ontology-based description) [23] or they can be manually defined for non-formal representations.

Since the Bank Security Booth scenario is completely modeled by means of a formally verifiable representation, interfaces between different domains are automatically generated and they are implemented as simple event-translation machines. Event-translation machines, or connectors, are single-state machines that map an incoming event to an outgoing event having a different name. Figure 8 shows the interface between the Actionable and the Controllable domains whereas Figure 9 depicts the glue layer between Controllable and AmI.

5.6. AmI

The bank security booth control algorithm is implemented as a state-chart controller, coherently with the modeling choices taken for the other representation domains. This design choice allows on one hand to get a fully abstract and easy to simulate model. On the other hand, state diagram modeling supports defining security constraints on the control algorithm in terms of temporal logic axioms that can subsequently be verified through formal model checking techniques, e.g., by exploiting
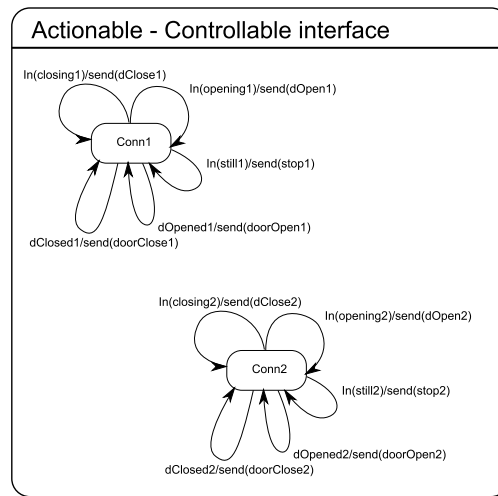

Figure 8: Interfaces between the Actionable and Controllable domains.
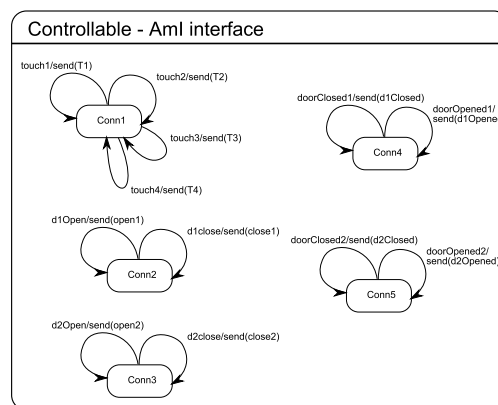

Figure 9: Interfaces between the Controllable and the AmI domains.

the UMC model checker [33]. Another advantage of state charts, when used as a model for the AmI Domain, is that the Home Gateway may interpret them directly: in this case the simulation of the AmI Domain becomes the real implementation of AmI functionalities, avoiding the step of coding its behavior and the otherwise-necessary correctness verification.
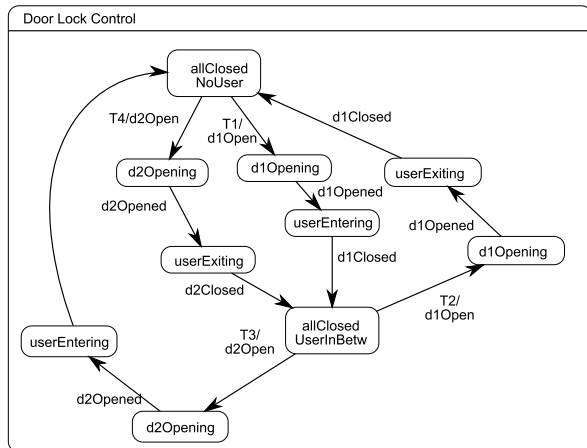

Figure 10: A sample Bank Security Booth control.

6. Bank Security Booth Verification

The Bank Security Booth use case described in the previous section has been implemented both as a fully simulated model (row 2 in Table 2, Section 4) by exploiting the DogSim framework [23] and as a partially emulated model with real touch sensors and door actuators (10th row of Table 2), by exploiting the EmuDog libraries [32].

Simulation experiment. State diagrams referred to every single domain components have been encoded in SCXML [39] and probabilistic transitions have been obtained by exploiting SCXML-Java integration provided by the Apache Commons SCXML engine[5], which is part of the DogSim simulation engine. In particular, probabilistic transitions have been implemented as event triggers exploiting an external Java function able to generate random numbers between 0.0 and 1.0, with a uniform distribution (Figure 11 shows an excerpt of the User domain state machine).

All domain models have been implemented in separated SCXML state machines and domain interfaces

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE scxml SYSTEM "template.dtd">
<!-- @device=bankUser -->
<scxml xmlns="http://www.w3.org/2005/07/scxml"
    version="1.0">
    <state id="&id;BankUserModel">
        <datamodel>
            <data name="&id;decisionProbability"/>
        </datamodel>
        <initial>
            <transition target="&id;UserOut"/>
        </initial>
        <state id="&id;UserOut">
            <onentry>
                <send sendid="&id;decideToEnter"
                    targettype="'scxml'"
                    event="'&id;decideToEnter'"
                    delay="tmGen.tDecide()"/>
                <assign name="&id;decisionProbability"
                    expr="rndGen.random()"/>
            </onentry>
            <!-- probabilistic transition: 0.5
                    remain in the decision state and
                    0.5 decide to enter -->
            <transition event="&id;decideToEnter"
                cond="&id;decisionProbability ge 0.5"
                target="&id;UserOut"/>
            <transition event="&id;decideToEnter"
                cond="&id;decisionProbability lt 0.5"
                target="&id;WaitingToEnter">
                <send targettype="'scxml'"
                    event="'&id;entryRequest'"/>
            </transition>
            <onexit>
                <cancel sendid="&id;decideToEnter"/>
            </onexit>
        </state>
        <state id="&id;WaitingToEnter">
            <onentry>
                <send sendid="&id;decideToLeave"
                    targettype="'scxml'"
                    event="'&id;decideToLeave'"
                    delay="tmGen.tDecide()"/>
                <assign name="&id;decisionProbability"
                    expr="rndGen.random()"/>
            </onentry>
            <!-- probabilistic transition: 20\% of
                possibilities to decide to leave and
                80\% probability to wait for entering
                the bank -->
            <transition event="&id;decideToLeave"
                cond="&id;decisionProbability ge 0.8"
                target="&id;UserOut"/>
            <transition event="&id;decideToLeave"
                cond="&id;decisionProbability lt 0.8"
                target="&id;WaitingToEnter"/>
            <transition event="&id;extDoorOpen"
                target="&id;InBetweenEntering">
                <send targettype="'scxml'"
                    event="'&id;entryRequest2'"
                    delay="tmGen.tStep()"/>
            </transition>
            <onexit>
                <cancel sendid="&id;decideToLeave"/>
            </onexit>
        </state>
        ....
    </state>
</scxml>
```

Figure 11: An excerpt of the User state machine in SCXML.

are created at runtime by instantiating suitable connector machine templates. The resulting environment machine is composed of 64 concurrent states and takes a negligible time[6] to be created. In order to enable visual inspection of simulation results, and to provide a rough, immediate overview of simulation evolution we developed a really simple graphical interface showing the user moving in and out of the bank security booth (Figure 12), as dictated by the state machine run-time evolution.
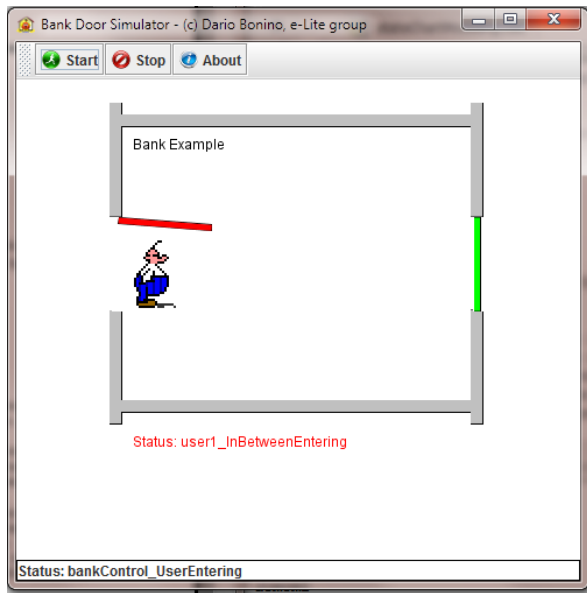

Figure 12: The Bank Security Booth simulation GUI.

Emulation experiment. In the mixed emulation environment (10th row of Table 2) touch sensor and door actuator models are replaced by real devices: 4 buttons (KNX[7]) and 2 door actuators (MyHome[8]). We exploit the DOG [38] gateway for accessing real domotic devices connected to two different home automation technologies: BTicino MyHome and KNX, respectively. Actions on the real buttons are converted into suitable state machines events by exploiting the ability of DOG to abstract and dispatch domotic events in a common, shared format named DogMessage. DogMessages are converted to DogSim events by means of the DOG2DogSim adapter bundled with the DogSim simulation framework. On the other way around, events generated by the simulated elements, e.g., by the bank se-

curity booth control, are converted back to the DogMessage format and, thanks to DOG, dispatched to the real actuators, which can then operate a real door. In our setting, we just limited the real part to buttons and actuators installed in 2 demo cases (shown in Figure 13), while door sensors are virtual, i.e., simulated through state charts.

Figure 13: The demo cases used for the emulation of the Bank Security Booth use case, and the Asus eeePC running DOG.

Formal verification experiment. The DoMAIns approach also allowed formal verification of some system properties of the Bank Security Booth. As detailed in [31], to which the reader is referred for further details, the Statecharts for the system description (generated by DogSim) and for the Security Booth controller have been verified by means of the UMC model checker [40]. More than 50 system properties have been modeled in the UCTL logic [41], and they have been proven true over the Statechart model. Properties included liveness (each request mush eventually be honored by the system), security (both doors may never be open at the same time), as well as correct internal protocol implementation (i.e., correct handshake between the system controller and door actuators).

7. Conclusions

This paper proposed DoMAIns: a *flexible* and *neat* modeling approach that defines an engineered methodology and a modeling framework to effectively design Intelligent (Domotic) Environments and Smart Home Automation systems.

DoMAIns exploits so-called *Representation Domains* to achieve modular and effective IDE design.

---

[6]less than 100 ms on an Intel P8400 (2.2GHz) processor.
[7]http://www.knx.org/
[8]http://www.myhome-bticino.it/

Modeling concerns and techniques are clearly identified and well separated, seamlessly supporting different representation solutions as well as hardware-in-the-loop simulation (emulation). A sample use case concerning two steps of the design of a simplified Bank Security Booth has been illustrated, showing how to apply the DoMAIns approach to solve real world design tasks and confirming the overall approach feasibility.

Clearly many efforts are still needed to fully realize the vision underlying the DoMAIns approach, however a first founding step has been accomplished, on top of which more effective and comprehensive IDE modeling can be devised.

Future work will target both semi-automatic generation of DoMAIns representations starting from DogOnt descriptions of IDEs, and formal model checking for statechart-based DoMAIns models. Both efforts are driven by the goal of closing the design loop supporting semi-automatic end-to-end modeling and verification for smart environments.

References

[1] V. Miori, D. Russo, M. Aliberti, Domotic technologies incompatibility becomes user transparent, Commun. ACM 53 (1) (2010) 153–157.

[2] E. Tokunaga, H. Ishikawa, M. Kurahashi, Y. Morimoto, T. Nakajima, A Framework for Connecting Home Computing Middleware, in: International Conference on Distributed Computing Systems Workshops (ICDCSW02), 2002, pp. 765–770.

[3] D. Zhang, T. Gu, X. Wang, Enabling Context-aware Smart Home with Semantic Technology, International Journal of Assistive Robotics and Mechatronics (formerly known as International Journal of Human-friendly Welfare Robotic Systems) 6 (4) (2005) 12–20.

[4] V. Miori, L. Tarrini, M. Manca, G. Tolomei, An Open Standard Solution for Domotic Interoperability, IEEE Transactions on Consumer Electronics 52 (2006) 97–103.

[5] P. Pellegrino, D. Bonino, F. Corno, Domotic House Gateway, in: Proceedings of the 2006 ACM symposium on Applied computing, 2006, pp. 1915–1920.

[6] D. J. Cook, J. C. Augusto, V. R. Jakkulaa, Ambient intelligence: Technologies, applications, and opportunities, Pervasive and Mobile Computing Volume 5, Issue 4 (2009) 277–298.

[7] M. Weiser, The computer for the twenty-first century, Scientific American September (1991) 94–100.

[8] M. McCullough, Digital Ground, Architecture, Pervasive Computing, and Environmental Knowing, MIT Press, 2004.

[9] D. J. Cook, G. M. Youngblood, E. O. H. III, K. Gopalratnam, S. Rao, A. Litvin, F. Khawaja, MavHome: An Agent-Based Smart Home., in: International Conference on Pervasive Computing and Communications, IEEE Computer Society, 2003, pp. 521–524.

[10] S. Helal, W. Mann, H. El-Zabadani, J. King, Y. Kaddoura, E. Jansen, The Gator Tech Smart House: A Programmable Pervasive Space, IEEE Computer 0018-9162/05 (2005) 64–74.

[11] S. Poslad, Ubiquitous Computing: Smart Devices, Environments and Interactions, Wiley, 2009.

[12] E. Meshkova, J. Riihijarvi, P. Mahonen, C. Kavadias, Modeling the home environment using ontology with applications in software configuration management, in: Proc. International Conference on Telecommunications ICT 2008, 2008, pp. 1–6.

[13] S. Runde, H. Dibowski, A. Fay, K. Kabitzsch, Integrated automated design approach for building automation systems, in: Proc. IEEE International Conference on Emerging Technologies and Factory Automation ETFA 2008, 2008, pp. 1488–1495.

[14] M. Jimenez, F. Rosique, P. Sanche, B. lvarez, A. Iborra, Habitation: A domain-specific language for home automation, IEEE SOFTWARE 26 (4) (2009) 30–38.

[15] L. Sommaruga, A. Perri, F. Furfari, DomoML-env: an ontology for Human Home Interaction, in: Proceedings of SWAP 2005, the 2nd Italian Semantic Web Workshop, Trento, Italy, December 14-16, 2005, CEUR Workshop Proceedings, 2005.

[16] D. Bonino, F. Corno, DogOnt - Ontology Modeling for Intelligent Domotic Environments, in: A. Sheth, S. Staab, M. Dean, M. Paolucci, D. Maynard, T. Finin, K. Thirunarayan (Eds.), International Semantic Web Conference, no. 5318 in LNCS, Springer-Verlag, 2008, pp. 790–803.

[17] A. Kofod-Petersen, A. Aamodt, Contextualised Ambient Intelligence through Case-Based Reasoning, in: T. R. Roth-Berghofer, M. H. Göker, H. A. Güvenir (Eds.), Proceedings of the Eighth European Conference on Case-Based Reasoning (ECCBR 2006), Vol. 4106 of Lecture Notes in Computer Science, Springer Verlag, Ölüdeniz, Turkey, 2006, pp. 211–225.

[18] D. Preuveneers, J. V. den Bergh, D. Wagelaar, A. Georges, P. Rigole, T. Clerckx, Y. Berbers, K. Coninx, V. Jonckers, K. D. Bosschere, Towards an extensible context ontology for Ambient Intelligence, in: Second European Symposium on Ambient Intelligence, Vol. 3295 of LNCS, Springer, Eindhoven, The Netherlands, 2004, pp. 148 – 159.

[19] H. C. T. Finin, A. Joshi, Ontologies for Agents: Theory and Experiences, Birkhuser Basel, Los Alamitos, CA, USA, 2005, Ch. The SOUPA Ontology for Pervasive Computing, pp. 233–258.

[20] G. Conte, D. Scaradozzi, A. Perdon, M. Cesaretti, G. Morganti, A simulation environment for the analysis of home automation systems, in: Proc. Mediterranean Conference on Control & Automation MED '07, 2007, pp. 1–8.

[21] G. Conte, D. Scaradozzi, Modeling and Control of Complex Systems, CRC Press, 2007, Ch. 15 - An Approach to Home Automation by means of MAS Theory.

[22] F. Klugl, A Validation Methodology for Agent-Based Simulations, in: R. Menezes, M. Viroli (Eds.), Symposium on Applied Computing, ACM Press, 2008, pp. 39–43.

[23] D. Bonino, F. Corno, Dogsim: A state chart simulator for domotic environments, in: Eighth Annual IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), 2010.

[24] J. Lertlakkhanakul, J. W. Choi, M. Y. Kim, Building data model and simulation platform for spatial interaction management in smart home, Automation in Construction 17 (2008) 948–957.

[25] R. Filman, T. Elrad, S. C. M.Aksit, Aspect Oriented Software Development, Addison Wesley Professional, 2004.

[26] A. Rashid, T. Cottenier, P. Greenwood, R. Chitchyan, R. Meunier, R. Coelho, M. Sudholt, W. Joosen, Aspect-oriented software development in practice: Tales from aosd-europe, Computer 43 (2010) 19–26.

[27] S. Propp, G. Buchholz, P. Forbrig, Task Model-Based Usability Evaluation for Smart Environments, in: P. Forbrig, F. Paterno(Eds.), HCSE/TAMODIA, Vol. 5247 of LNCS, IFIP International Federation for Information Processing, Springer, 2008, pp. 29–40.

[28] W. Maik, S. Propp, P. Forbrig, HCI-Task Models and Smart

Environments, in: A. M. P. Peter Forbrig, Fabio Paternò(Ed.), Human-Computer Interaction Symposium, Vol. 272, IFIP International Federation for Information Processing, Springer, 2008, pp. 21–32.

[29] D. Bonino, F. Corno, Rule-based intelligence for domotic environments, Automation in Construction 19 (2010) 183–196.

[30] F. Corno, S. Muhammad, Formal verification of device state chart models, in: IE'11, The 7th International Conference on Intelligent Environments,, 2011. doi:10.1109/IE.2011.36.

[31] F. Corno, S. Muhammad, Design time methodology for the formal verification of intelligent domotic environments, in: P. Novais, D. Preuveneers, J. Corchado (Eds.), Ambient Intelligence - Software and Applications, Vol. 92 of Advances in Intelligent and Soft Computing, Springer Berlin / Heidelberg, 2011, pp. 9–16. doi:10.1007/978-3-642-19937-0_2.

[32] D. Bonino, F. Corno, Modeling, simulation and emulation of intelligent domotic environments, Automation in Construction In Press. doi:10.1016/j.autcon.2011.03.014.

[33] M. H. ter Beek, F. Mazzanti, S. Gnesi, CMC-UMC: A framework for the verification of abstract service-oriented properties, in: SAC '09: Proceedings of the 2009 ACM symposium on Applied Computing, ACM Press, 2009, pp. 2111–2117.

[34] D. N. Jansen, H. Hermanns, J.-P. Katoen, A probabilistic extension of uml statecharts: Specification and verification, in: Intl. Symp. on Formal Techniques in Real-Time and Fault-Tolerant Systems, LNCS 2469, Springer, 2002, pp. 355–374.

[35] H. A. Hansson, Time and probability in formal design of distributed systems, Ph.D. thesis, University of Uppsala (1991).

[36] M. L. Puterman, Markov Decision Processes: Discrete Stochastic Dynamic Programming, Wiley, New York, 1994.

[37] H. Hansson, B. Jonsson, A logic for reasoning about time and reliability, Formal Aspects of Computing 6 (1994) 102–111.

[38] D. Bonino, E. Castellina, F. Corno, The DOG Gateway: Enabling Ontology-based Intelligent Domotic Environments, IEEE TRANSACTIONS ON CONSUMER ELECTRONICS 54/4 (2008) 1656–1664.

[39] J. Barnett, R. Akolkar, R. Auburn, M. Bodell, D. C. Burnett, J. Carter, S. McGlashan, State Chart XML (SCXML): State Machine Notation for Control Abstraction, Tech. rep., W3C Working Draft (2009).

[40] S. Gnesi, F. Mazzanti, On the fly model checking UML State Machines, in: ACIS International Conference on Software Engineering Research, Management and Applications, 2004, pp. 331 – 3382.

[41] F. Mazzanti, UMC 3.3 User Guide, ISTI Technical Report 2006-TR-33, ISTI-NNR Pisa-Italy (September 2006).