Fault-detection in Ambient Intelligence based on the modeling of physical effects.

Ahmed Mohamed

Ce projet est cofinancé par l'Union européenne. L'Europe s'engage en Ile-de-France avec le Fonds européen de développement régional

N° d'ordre : 2013-23-TH

SUPÉLEC

ÉCOLE DOCTORALE STITS

« Sciences et Technologies de l'Information, des Télécommunications et des Systèmes »

THÈSE DE DOCTORAT

DOMAINE : STIC
Spécialité : Informatique

Soutenue le
19 novembre 2013

par :

Ahmed MOHAMED

Détection de défaillances fondée sur la modélisation des effets physiques dans l'ambiant

-

Fault detection in Ambient Intelligence based on the modeling of physical effects

Composition du jury :

Directeur de thèse :	**M. Yacine Bellik**	Univ. Paris-Sud, LIMSI-CNRS
Co-encadrant :	**M. Christophe Jacquet**	Supélec
Rapporteurs :	**M. Amar Ramdane-Cherif**	Univ. Versailles
	M. Hani Hagras	Univ. Essex
Examinateurs :	**M. Christophe Kolski**	Univ. Valenciennes
	M. Nicolas Sabouret	Univ. Paris-Sud, Supéle
Membre invité :	**M. Bruno Jean-Bart**	Trialog

This work has been performed within the CBDP project, a project co-funded by the European Union. Europe is involved in Région Île-de-France with the European Regional Development Fund.

Abstract

This thesis takes place in the field of Ambient Intelligence (AmI). Ambient Intelligent Systems are interactive systems composed of many heterogeneous components. From a hardware perspective these components can be divided into two main classes: sensors, using which the system observes its surroundings, and actuators, through which the system acts upon its surroundings in order to execute specific tasks.

From a functional point of view, the goal of Ambient Intelligent Systems is to activate some actuators, based on data provided by some sensors. However, sensors and actuators may suffer failures. Our motivation in this thesis is to equip ambient systems with self fault-detection and diagnosis capabilities allowing them to check autonomously whether the intended actions were performed correctly by the actuators.

To address this issue, one could apply classical control theory to pre-determine closed control loops using the available sensors. However, the particularity of ambient systems is that instances of physical resources (mainly sensors and actuators) are not necessarily known at design time; instead they are dynamically discovered at run-time. In consequence, such control loops cannot be pre-determined.

We propose an approach in which the fault detection and diagnosis in ambient systems is dynamically done at run-time, while decoupling actuators and sensors at design time. We introduce a Fault Detection and Diagnosis framework modeling the generic characteristics of actuators and sensors, and the effects that are expected on the physical environment when a given action is performed by the system's actuators. These effects are then used at run-time to link actuators (that produce them) with the corresponding sensors (that detect them). Most importantly the mathematical model describing each effect allows the calculation of the expected readings of sensors. Comparing the predicted values with the actual values provided by sensors allow us to achieve fault-detection in dynamic and heterogeneous ambient systems.

Résumé

Cette thèse s'inscrit dans le domaine de l'intelligence ambiante (Ambient Intelligence - AMI). Les systèmes d'intelligence ambiante sont des systèmes interactifs composés de plusieurs éléments hétérogènes. D'un point de vue matériel, les composants de ces systèmes peuvent être divisés en deux catégories principales : les capteurs, que le système utilise pour observer son environnement, et les effecteurs, à travers lesquels le système agit sur son environnement afin d'exécuter des tâches spécifiques.

D'un point de vue fonctionnel, l'objectif des systèmes d'intelligence ambiante est d'activer certains effecteurs, sur la base des mesures réalisées par des capteurs. Toutefois, les capteurs et les effecteurs peuvent subir des défaillances. Notre motivation dans cette thèse est de munir les systèmes ambiants de capacités d'auto-détection des pannes, pour leur permettre de vérifier de manière autonome si les actions prévues ont été effectuées correctement par les effecteurs.

Pour résoudre ce problème, on pourrait appliquer des techniques classiques en automatique, et ainsi prédéterminer des boucles de régulation ad-hoc utilisant les capteurs disponibles. Cependant, une particularité des systèmes ambiants est leur ouverture : les ressources physiques (principalement les capteurs et effecteurs) ne sont pas nécessairement connues au moment de la conception, mais elles sont plutôt découvertes dynamiquement lors de l'exécution. En conséquence, ces boucles de régulation ne peuvent pas être établies à l'avance.

Nous proposons une nouvelle approche dans laquelle la stratégie de détection de défaillances dans un système ambiant est déterminée dynamiquement lors de l'exécution. Pour cela, les couplages entre capteurs et effecteurs ne sont pas déterminés par le concepteur du système, mais déduits automatiquement lors de l'exécution. Ceci est rendu possible par la modélisation des caractéristiques des capteurs, des effecteurs, ainsi que des phénomènes physiques (que nous appelons effets) qui sont attendus dans l'environnement ambiant quand une action donnée est effectuée par un effecteur. Ces effets sont utilisés lors du fonctionnement du système pour lier les effecteurs (produisant les effets) avec les capteurs correspondants (détectant les effets).

Nous introduisons une plateforme de détection des pannes qui génère à l'exécution un modèle de prédiction des valeurs attendues sur les capteurs. Ce modèle, de nature hétérogène (il mêle flots de données et automates finis) est exécuté par un outil adapté (ModHel'X) de façon à fournir les valeurs attendues à chaque instant. Notre plateforme compare alors ces valeurs avec les valeurs réellement mesurées de façon à détecter les défaillances.

Acknowledgments

Foremost, I would like to express my sincere gratitude to my thesis advisor Assistant Professor Yacine Bellik and my thesis supervisor Associate Professor Christophe Jacquet for their support, patience, and immense knowledge. Their encouragements and advices were necessary for me to proceed through with my research works and to complete my dissertation.

Besides my advisors, I would like to thank the rest of my thesis committee: Professor Kolski Christophe, Professor Ramdane-Cherif Amar, Professor Hagras Hani, Professor Nicolas Sabouret, and "Trialog" Manager Bruno Jean-Bart for accepting to be in my thesis committee and for their insightful comments and questions.

Special thanks go to the members and fellow lab mates at the departments of computer science at Supelec and at Limsi for their friendship and assistance.

I also wish to thank my parents, to whom I owe everything, my brother, and my friends for their unconditional love and support.

Table of Contents

List of Figures

List of Tables

Chapter 1:

Summary of the Thesis in French

Chapter 1.

Summary of the Thesis in French

1.1. Introduction

1.1.1. Contexte et Motivations

Nos travaux s'inscrivent dans le domaine de l'intelligence ambiante (Ambient Intelligence - AMI). Les systèmes d'intelligence ambiante sont des systèmes interactifs composés de plusieurs éléments hétérogènes qui peuvent être catégorisés dans deux classes principales : les capteurs, que le système utilise pour observer son environnement, et les effecteurs, par lesquels le système agit sur son environnement pour exécuter des tâches particulières. Le but de ces tâches est d'assurer le confort de l'occupant de l'environnement, faciliter la réalisation de certains travaux, superviser et aider les utilisateurs qui ont besoin d'aide ou de soins. Les systèmes d'intelligence ambiante s'appliquent à de nombreux contextes d'utilisation : usage personnel (« maisons intelligentes »), cadre professionnel (hôpitaux par exemple), ou cadre industriel (usines).

De nombreux aspects du domaine de l'intelligence ambiante font l'objet de travaux de recherche et de développement, comme le matériel et les technologies déployés dans les environnements ambiants, les concepts et les techniques qui ajoutant une couche d'intelligence à l'environnement ambiant, la contextualisation et les techniques d'auto-configuration, la modélisation de tâches, etc. Il y a principalement trois types de tâches qui sont exécutées dans un environnement d'intelligence ambiante : des tâches effectuées par les utilisateurs (très difficiles à prévoir), des tâches effectuées par le système (via les effecteurs), et des tâches exécutées dans le cadre des interactions homme-machine. Dans le domaine de l'intelligence ambiante, ces tâches sont abordées et étudiés selon des différents angles (voir **Figure 1**).

Notre travail porte sur les tâches effectuées par le système via des effecteurs. Notre objectif est de superviser les actions du système afin de détecter tout dysfonctionnement. Outre le fait que les effecteurs sont sujets à des défaillances, l'hétérogénéité des divers dispositifs des systèmes d'intelligence ambiante rend difficile la détection de la cause réelle d'une panne. Des thèmes tels que l'adaptativité des équipements, l'auto-résolution de problèmes et la tolérance aux pannes sont ainsi étudiés dans le domaine. Nous estimons qu'un système ambiant doit pouvoir « découvrir », « comprendre » et « corriger » les défauts qui se produisent durant l'exécution de ses tâches d'une manière autonome. La « découverte » des défauts est appelée détection de défaut, et la « compréhension » des défauts (déduire plus d'informations au sujet d'une panne, y compris son origine idéalement) est appelée diagnostic.

L'objectif général de notre travail est de proposer une architecture générique pour le diagnostic de pannes dans les systèmes d'intelligence ambiante. Nous détaillons et validons, à travers des exemples, la partie relative à la détection des défauts.

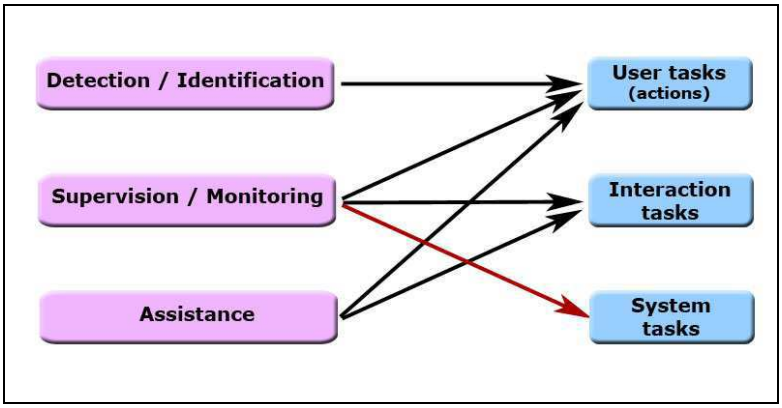

Figure 1. Principaux axes de recherche étudiant les tâches dans un environnement ambiant

Un système ambiant est souvent en interaction avec son environnement. Pour cela le système perçoit l'état de son environnement en utilisant des capteurs, et agit en conséquence sur l'environnement en utilisant des effecteurs. D'une part, pour assurer la réalisation de ses objectifs, le système ambiant dépend fortement de la bonne exécution des tâches qui sont effectuées par ses effecteurs. D'autre part, les systèmes d'intelligence ambiante sont conçus pour maintenir un certain niveau de non-ingérence, afin d'éviter toute gêne des usagers. C'est pourquoi les tâches sont généralement exécutées en arrière-plan d'une manière non perceptible par les utilisateurs. Une telle exigence rend inacceptable le fait d'inonder l'utilisateur avec un grand nombre de données relatives à la détection de pannes. En revanche, ne pas informer les utilisateurs d'un défaut détecté peut être dangereux, car les personnes pourraient continuer à se reposer sur un service défaillant **[1]**. Ceci peut être particulièrement problématique pour des applications critiques comme garantir la sûreté des patients dans un hôpital.

Pour ces raisons, nous voulons donner aux systèmes ambiants le moyen de vérifier de manière autonome si oui ou non les tâches systèmes ont été effectuées correctement. Quand un système ambiant envoie des ordres à un effecteur, la bonne façon de vérifier si un ordre a été exécuté correctement est d'exploiter les données des capteurs afin de s'assurer que l'état de l'environnement a changé comme prévu. Prenons l'exemple d'un système qui allume une ampoule. L'infrastructure matérielle et les moyens de communication peuvent permettre au système de vérifier si la commande a été correctement transmise et que le circuit électrique de l'ampoule a été fermé. Cependant, de nombreux facteurs pourraient empêcher la lumière d'être émise, par exemple l'ampoule pourrait avoir été endommagée et donc ne pas s'allumer, ou bien elle pourrait être recouverte par un objet opaque, etc. Donc, pour vérifier que la lumière est vraiment émise, il faut utiliser un capteur de lumière (voir **Figure 2**). Une solution classique, issue du domaine de l'automatique consiste à installer des capteurs ad-hoc pour réaliser des boucles de régulation. Cependant, l'une des principales particularités de systèmes ambiants est que, contrairement aux systèmes traditionnels, les ressources physiques (principalement les capteurs et effecteurs) ne sont pas nécessairement connues au moment de la conception du système. Elles sont dynamiquement découvertes et peuvent apparaître et/ou disparaître au moment de l'exécution, donc on peut difficilement installer des capteurs ad-hoc et prédéterminer des boucles de régulation.

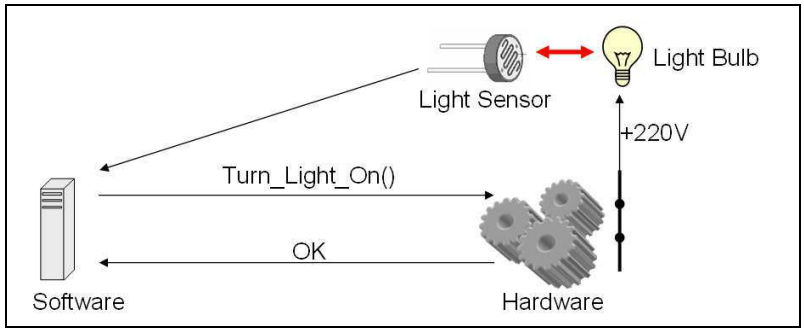

Figure 2. Boucle de régulation ad-hoc classique

Dans cette thèse, nous proposons une solution qui permet la construction dynamique des liens déduits entre les effecteurs et capteurs dans les systèmes ambiants en exploitant les ressources disponibles à un moment donné, et l'utilisation de ces liens pour détecter l'existence de défaillances en cours d'exécution. L'approche est basée sur la modélisation des phénomènes physiques (que nous appelons les effets) qui se produisent dans l'environnement quand un effecteur donné est activé. Les effets sont caractérisés par des lois physiques qui peuvent être modélisées selon différents niveaux de détails. Ces lois dépendent des paramètres physiques qui sont associés aux effecteurs et aux capteurs. En exploitant la connaissance sur les capteurs et effecteurs présents à un moment donné ainsi que ces les lois physiques, le système est capable de créer automatiquement des associations entre des capteurs et des effecteurs. Ensuite, en effectuant les calculs appropriés, le système déduit les mesures attendues au niveau d'un capteur donné lorsqu'une certaine action est effectuée par un effecteur (par exemple, une augmentation de la température peut être prévue après un certain laps de temps quand un système de chauffage est activé). De cette façon, le système est capable, par la comparaison de ces valeurs calculées avec les mesures réelles des capteurs, de détecter les défaillances et éventuellement d'utiliser les données collectées pour produire un diagnostic. La détection de pannes se fait au moment de l'exécution, sans nécessiter le couplage explicite de capteurs et d'effecteurs au moment de la conception. Nous estimons que cette technique est bien adaptée au caractère ouvert des systèmes d'intelligence ambiante, dans lesquels on peut ajouter et retirer des composants en cours d'exécution.

1.1.2. Plan de la thèse

Cette thèse est organisée comme suit. Le chapitre III est un état de l'art, dans lequel nous introduisons le domaine de l'intelligence ambiante. Nous présentons également quelques travaux relatifs à la détection et au diagnostic des pannes dans le domaine de l'automatique. Nous analysons les limites de ces approches dans le cadre d'une mise en œuvre dans des environnements ambiants. Le chapitre IV détaille notre approche de détection et de diagnostic de pannes. Nous décrivons l'architecture de notre plateforme, les modèles qui la composent, la structure de ces modèles, leur hiérarchie. Nous expliquons aussi comment ces modèles nous permettent de réaliser la détection et le diagnostic de pannes dans un environnement d'intelligence ambiante. Le chapitre V est consacré à la mise en œuvre de notre plateforme. Nous détaillons l'implémentation de notre plateforme en Java et en utilisant ModHel'X, une plateforme permettant la représentation et l'exécution de modèles hétérogènes. Nous abordons également l'intégration de notre plate-forme dans une application de « Ambient Assisted Living », et la mise en œuvre d'un simulateur permettant de dérouler des scénarios. Dans le chapitre VI, nous utilisons notre simulateur pour tester la plateforme sur des scénarios plus réels. Le chapitre VII est une conclusion dans laquelle nous discutons des apports et des limites de notre approche. Nous donnons des pistes pour surmonter ces limitations et améliorer les performances de notre approche et la précision de la détection des pannes et du diagnostic.

1.2. Etat de l'art

Depuis que l'on construit et utilise des machines, assurer le bon fonctionnement de ces machines est une problématique importante. La détection des pannes et leur diagnostic est un domaine de recherche à part entière, qui s'est complexifié à mesure de la complexification des systèmes **[2]**. L'idée générale est de créer des modèles représentant le système diagnostiqué, et de superposer ces modèles avec le système réel afin de détecter des pannes. Parmi les systèmes qui peuvent bénéficier de la détection de défaillances, nous nous concentrons dans cette thèse sur les systèmes d'intelligence ambiante.

1.2.1. Intelligence Ambiante (AmI)

" Les technologies les plus importantes sont celles qui disparaissent. Elles empreignent la vie quotidienne jusqu'à se fondre en elle."- M. Weiser **[22]**

L'intelligence ambiante (AmI) est une vision du monde futur, où la technologie est omniprésente mais invisible. Les utilisateurs ne sont pas nécessairement conscients d'interagir avec des environnements très riches technologiquement. En effet, un système d'intelligence ambiante est un système interactif dans lequel les capacités de traitement et d'interaction sont dissimulées dans les outils du quotidien, facilitant ainsi l'introduction d'une couche d'intelligence faisant de l'ensemble un environnement intelligent.

Les maisons intelligentes, les hôpitaux intelligents, les transports publics intelligents et les usines intelligentes sont quelques exemples d'application des environnements intelligents. Les buts de ces applications varient de la simple facilitation des tâches de la vie quotidienne au contrôle et à la garantie de la sûreté des patients dans des hôpitaux.

Dans ce contexte les tâches de détection et diagnostic de pannes sont importantes, et méritent un traitement spécifique. En effet les systèmes ambiants présentent un certain nombre de caractéristiques particulières par rapport aux autres systèmes, rendant inefficaces l'application des approches de diagnostic classiques.

Parmi les caractéristiques spécifiques des systèmes ambiants, on trouve notamment leur aspect ouvert : des composants d'un système ambiant peuvent être ajoutés ou retirés au cours de l'exécution. Dans le cadre de la détection et du diagnostic de pannes, cette caractéristique rend impossible la prédétermination de boucles capteur-effecteur.

Les systèmes d'intelligence ambiante présentent d'autres caractéristiques particulières. Par exemple dans **[18]**, ils sont décrits comme des systèmes sensibles (ils détectent la présence des utilisateurs) et réactifs (ils réagissent à la présence des utilisateurs). Dans **[24]**, les auteurs insistent sur la transparence des services rendus par les systèmes ambiants : ils sont non intrusifs et disparaissent même à l'arrière-plan **[25]**.

Une autre, très importante, caractéristique des systèmes d'intelligence ambiante, est la gestion autonome **[29]**. Cela recouvre des propriétés telles que l'auto-configuration, l'auto-adaptation, l'auto-optimisation, l'auto-protection et l'auto-réparation. Cette dernière est possible après l'auto-diagnostic **[31]**.

1.2.2. Technologies

Nous présentons ici quelques technologies parmi celles qui sont les plus utilisées dans le contexte des environnements ambiants.

1.2.2.1. **Contrôleurs**

Les contrôleurs sont utilisés pour contrôler le fonctionnement des effecteurs. Il existe de simples contrôleurs permettant des fonctionnalités comme allumer/étendre (ou ouvrir/fermer) des effecteurs, comme il existe des contrôleurs plus avancés permettant un contrôle plus avancé de l'état des effecteurs (comme les variateurs de lumière ou les thermostats).

1.2.2.2. **Effecteurs**

Les effecteurs sont le moyen à travers lesquels le système ambiant agit sur son environnement. Les effecteurs permettent de convertir des ordres électriques en des actions physiques.

Dans cette thèse nous ne distinguons pas entre effecteurs et contrôleurs. Dans nos modèles nous considérons l'ensemble en tant qu'une seule entité que nous appelons effecteur.

1.2.2.3. **Capteurs**

Pour effectuer les bonnes actions au bon moment, le système ambiant a besoin d'être au courant de l'état de son environnement et d'être alerté de tout événement qui s'y produit.

Un capteur convertit une propriété physique mesurable en un signal pouvant être traité par le système.

1.2.3. Détection et Diagnostic de Pannes

Dans le domaine de l'automatique, les tâches de détection et diagnostic de pannes peuvent être divisées en trois catégories **[105]** :

- Détection de pannes : la détection, suite à une comparaison, d'une différence entre le système et son modèle.
- Isolement de la panne : après analyse des symptômes de la panne, on essaie de trouver la (les) cause(s) exacte(s) de la panne.
- Identification de la panne : on tente de déduire d'avantages d'informations à propos de la panne : son ampleur, son type, etc.

Le terme *diagnostic* recouvre l'isolement et l'identification de pannes.

Plusieurs travaux ont visé à appliquer des techniques de diagnostic à des systèmes ambiants. Par exemple dans **[143]** un middleware supervise constamment le contexte du système ambiant, afin de déclencher un ensemble de règles en fonction du contexte. Le système de diagnostic de ce middleware cherche à vérifier que les bonnes règles ont été déclenchées par le middleware, en supposant que les données fournies en entrée du moteur de contexte sont correctes. Dans **[145][146]**, on vérifie au contraire si ces données d'entrée sont bel et bien correctes.

D'autres projets ont visé à créer une infrastructure pour les systèmes d'intelligence ambiante, comme le Context Toolkit **[147]**, Aura **[148]**, Solar **[149]**, ConFab **[150]**, et Gaia **[151]**. Ces infrastructures fournissent des mécanismes de niveau système pour superviser des composants de l'application afin de faciliter la résolution de certains problèmes particuliers qui peuvent survenir au cours de son fonctionnement.

1.2.4. Architecture générale d'un environnement ambiant

Sur la **Figure 3** on peut observer l'architecture générale d'un environnement ambiant typique. L'architecture est centrée autour de l'utilisateur.

L'utilisateur peut interagir avec le système soit directement via les interfaces homme-machine disponibles, soit indirectement en agissant sur l'état et l'environnement. Dans ce dernier cas, le système ambiant peut détecter ces changements via des capteurs.

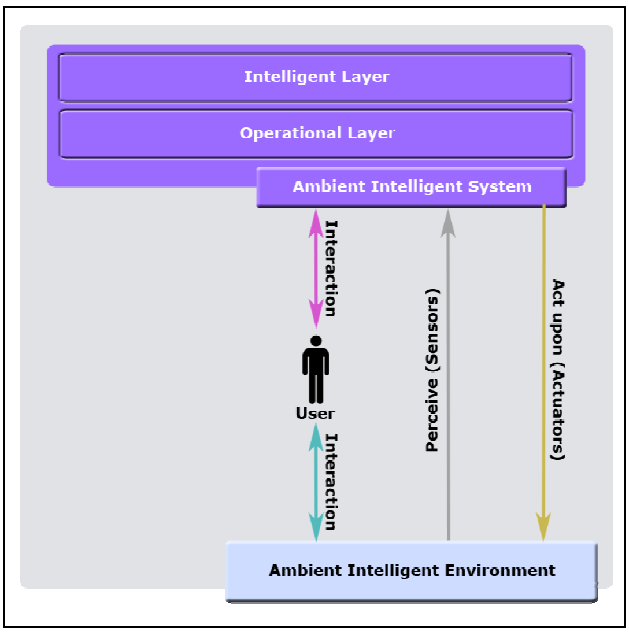

Figure 3. Architecture d'un environnement ambiant

1.3. AmILoop : Une plateforme de détection de pannes dans l'ambiant

Comme présenté sur la **Figure 4** (à gauche), nous utilisons trois niveaux de modèles. Un modèle abstrait universel est concrétisé en un modèle propre à chaque type d'environnement. Ce modèle concret est finalement instancié pour représenter les composants réels du système.

Le modèle abstrait est détaillé sur la **Figure 5**. Le modèle abstrait décrit l'environnement d'une manière qui découple les capteurs des effecteurs. Ceci est réalisé via le concept d'**Effet**, qui est une modélisation des conséquences physiques des actions des effecteurs sur l'environnement.

Le modèle concret hérite du modèle abstrait sa structure générale tout en définissant les types des composants, les phénomènes physiques attendus, les lois physiques les décrivant, et les liens entre toutes ces entités.

Les instances sont créées au moment de l'exécution par le moteur de contexte. Les instances représentent les composants actuellement présents ainsi que les valeurs actuelles des phénomènes physiques observés.

Ces données sont utilisées par le moteur de prédiction afin de prédire les mesures attendues au niveau des capteurs. La détection de pannes se fait par une comparaison entre valeurs

théoriques et valeurs mesurées. Les pannes potentielles sont ensuite diagnostiquées en utilisant un modèle de diagnostic (à droite).

1.3.1. Architecture générale d'AmILoop

1.3.1.1. **Point de vue structurel**

De point de vue structurel, les modèles utilisés par la plateforme AmILoop pour réaliser les tâches de détection et de diagnostic de pannes peuvent être regroupés en trois grands modèles (voir **Figure 6**):

- Le modèle statique de l'environnement : il est défini par le concepteur du système.
- Le modèle dynamique de l'environnement : il contient les types et instances réels.
- Le modèle de diagnostic : contenant les informations pour pouvoir isoler les pannes détectées. La nature de ce modèle dépend du type du moteur de diagnostic choisi (par exemple utilisation de moteur de raisonnement avec une ontologie comme modèle de diagnostic). Nous ne mettons pas de restriction sur les types de modèles possibles.

Sur la **Figure 6**, la relation « use » (utilise) définit le sens d'échange d'information entre modèles.

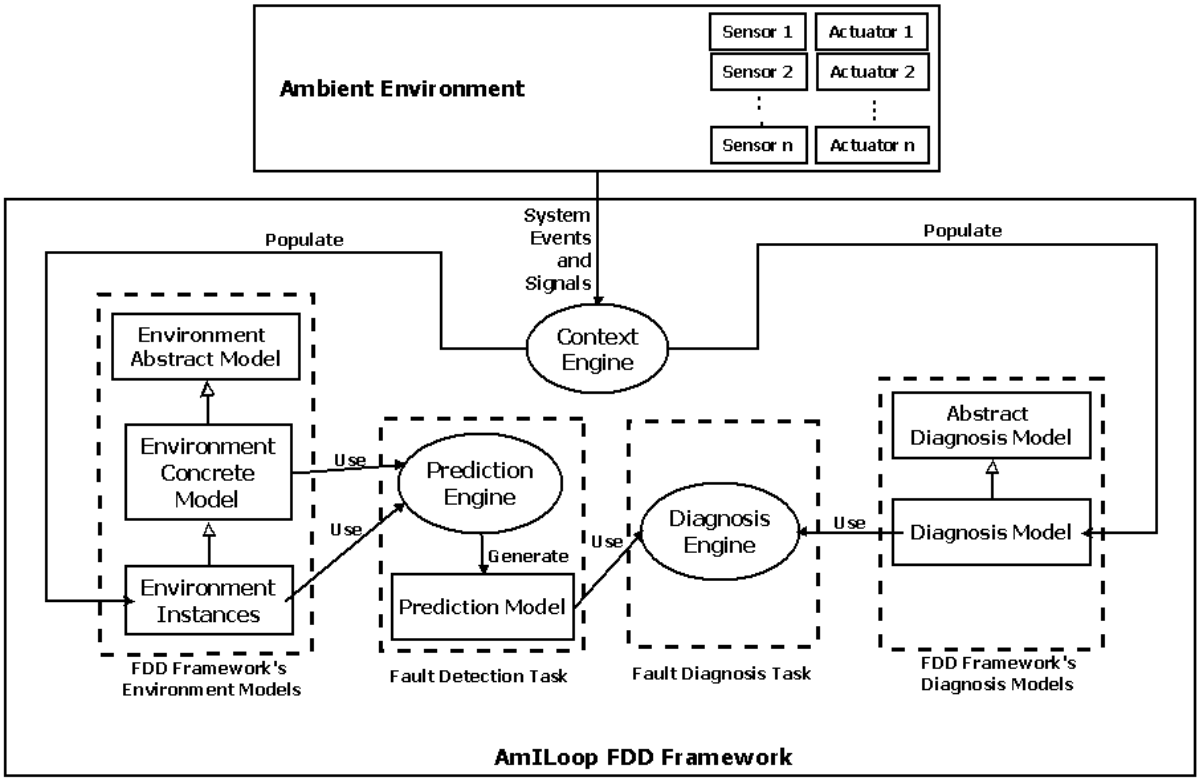

Figure 4. Architecture AmILoop dans le contexte d'un environnement ambiant

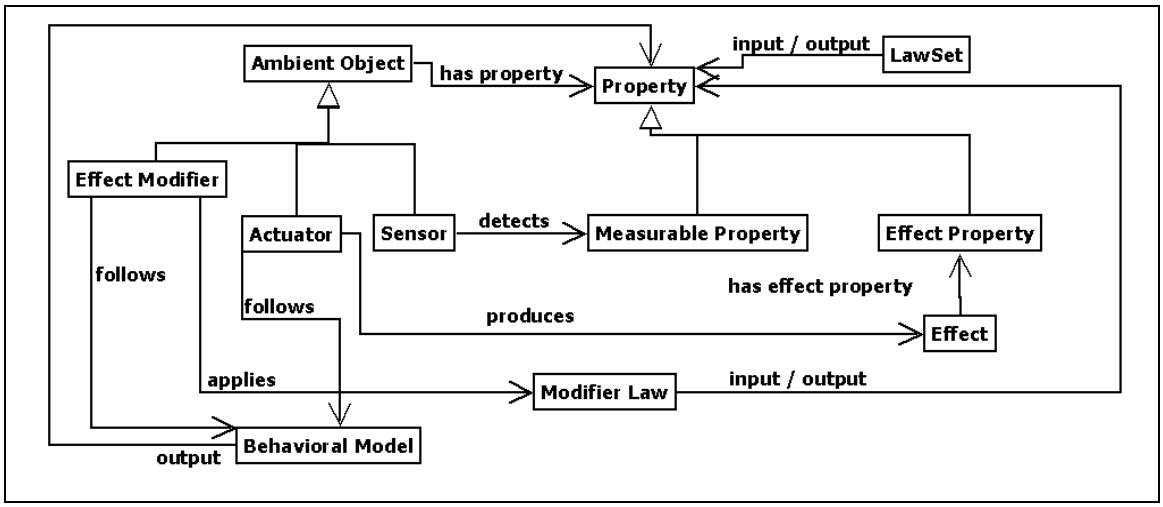

Figure 5. Le modèle abstrait d'AmILoop

1.3.1.2. **Point de vue comportemental**

Les tâches de détection de pannes et de diagnostic de pannes dépendent de modèles spécifiques. Ces modèles sont exploités par des moteurs correspondants afin de tirer des conclusions vis-à-vis des pannes. L'architecture d'exécution de la plateforme AmILoop (voir **Figure 51**) peut être résumée dans ces étapes :

i) Le **moteur de contexte (Context Engine)** utilise les informations de la couche matérielle et des informations du modèle statique (comme illustré sur la **Figure 7**) pour instancier les objets actuellement présents dans l'environnement et pour initialiser les valeurs des attributs de ces objets (comme les mesures des capteurs, les positions des objets, etc.).

ii) Les informations du modèle statique et les informations du modèle dynamique sont exploitées par le moteur de prédiction pour construire un modèle de prédiction. Ce dernier contient des modèles comportementaux de certains objets et des formules mathématiques permettant le calcul des valeurs attendues sur les capteurs. Le modèle de prédiction est à la base de la phase de détection de pannes.

iii) Les conclusions de la comparaison entre les valeurs théoriques des mesures des capteurs déduites par le modèle de prédiction et les valeurs réelles, les liens déduits entre capteurs et effecteurs, les états des composants, etc., sont ajoutés aux informations dans le moteur de diagnostic. Ces informations sont exploitées par le moteur de diagnostic pour déduire davantage d'informations sur les causes probables des pannes détectées. C'est la phase de diagnostic de pannes. Il est à noter que, dans cette thèse, nous proposons une plateforme qui met en œuvre les modèles et moteurs relatifs à la détection de défaillances. Cependant notre plateforme prévoit dans son architecture (sans l'implémenter) la tâche de diagnostic des pannes détectées.

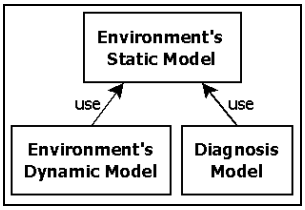

Figure 6. Les modèles d'AmILoop

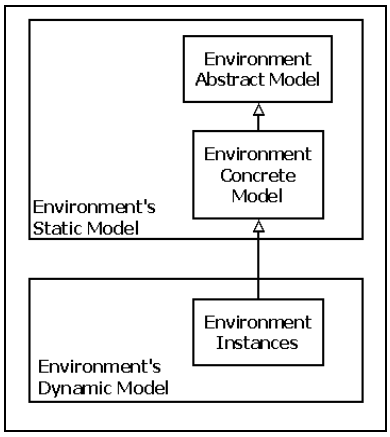

Figure 7. Hiérarchie des modèles d'AmILoop

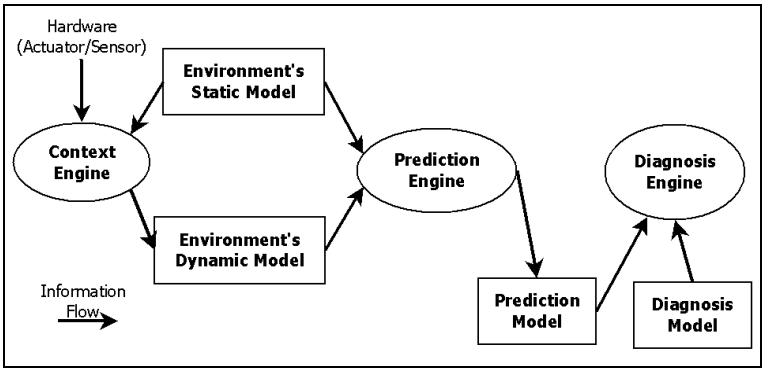

Figure 8. Architecture d'exécution d'AmILoop

1.3.1.3. **Découplage entre effecteurs et capteurs**

Pour pouvoir effectuer les tâches de détection de pannes et diagnostic dans un cadre ouvert tel qu'un environnement ambiant, dans lequel les composants ne sont pas forcement connus au moment de la conception du système et peuvent être découverts au moment de l'exécution, il est impossible d'exiger au moment de la conception un couplage explicite entre effecteurs et capteurs. Pour pouvoir faire le lien entre ces entités tout en les découplant, on introduit le concept d'effet, comme illustré sur la **Figure 5**. Les effecteurs sont des producteurs d'effets physiques, et les capteurs sont des récepteurs de ces effets. Des formules mathématiques modélisant les lois physiques permettent de relier les grandeurs produites par les premiers aux grandeurs mesurées par les seconds. Via les effets et en appliquant les formules au moment de l'exécution, les liens entre capteurs et effecteurs peuvent être déterminés dynamiquement et automatiquement. Cette approche permet en outre de modéliser un effecteur qui produit plus d'un seul effet physique. Par exemple une fenêtre agit à la fois sur la lumière et la température d'une pièce.

Notre proposition est donc bien adaptée au caractère ouvert des systèmes ambiants : les composants réels ne sont pas connus au moment de la conception ; ils ne sont découverts qu'au cours de l'exécution, et peuvent être ajoutés ou retirés à tout moment.

1.3.2. Concept d'effet

Un effet est une définition de la conséquence physique de l'action des effecteurs sur l'environnement. L'effet constitue le seul lien entre les capteurs et les autres composants (effecteurs principalement).

Le phénomène physique défini par l'effet est décrit par un ensemble de formules mathématiques (law-set). Ces formules utilisent comme paramètre de calcul les propriétés de l'effet (ex : intensité lumineuse) ainsi que les propriétés des composants (ex : position).

Les effets peuvent être modélisés selon différents niveaux de détails. Le choix de ce dernier est laissé au concepteur final. Le choix peut se baser sur :

- le contexte d'utilisation global. Par exemple un concepteur de système peut choisir une définition très détaillée de l'effet physique du propagation du son dans un environnement ambiant pour les malentendants.
- le contexte d'exécution courant. On peut imaginer un système programmé pour utiliser une définition détaillée de l'effet lumière pendant la nuit (là où les effecteurs de lumière sont les plus utilisés), et une définition moins détaillée (présence/absence) pendant la journée (détectant ainsi des erreurs dans l'état « ouvert ou fermé » des rideaux par exemple).

Par conséquent à chaque effet on peut associer une hiérarchie de law-sets du plus détaillé au moins détaillé.

Dans la suite nous détaillons l'effet lumière et nous donnons le modèle mathématique associé.

1.3.2.1. **Effet lumière**

Nous appelons effet lumière les ondes lumineuses produites par une source lumière. L'effet lumière est caractérisé par l'intensité du flux lumineux (en lumen). La propriété physique observée per les capteurs est l'intensité lumineuse (en lux).

Le modèle mathématique permet de prédire des valeurs théoriques des mesures des capteurs qui captent un effet particulier. On peut donner des formules (law-sets) à différents niveaux de détails. Pour l'effet lumière, on peut proposer par exemple trois modèles (du plus détaillé au moins détaillé) : prise en compte de l'intensité lumineuse en 3D, prise en compte de l'intensité en 2D, prise en compte simplement de la présence ou de l'absence de lumière. Dans ce qui suit nous décrivons le modèle mathématique de l'effet lumière dans un environnement 2D :

Nous appelons cet ensemble de lois *2DLightLawSet* ; il est composé des lois suivantes :

$$\text{sameZone}(s,a) = zone(s) \equiv zone(a) \qquad \textbf{(L1)}$$

$$\text{distance}(s,a) = \begin{cases} \sqrt{(x(s)-x(a))^2 + (y(s)-y(a))^2} & \textit{when } \text{sameZone}(s,a) \textit{ is true} \\ +\infty \quad \textit{when } \text{sameZone}(s,a) \textit{ is false} \end{cases} \qquad \textbf{(L2)}$$

$$\text{directLightExposure}(s,a) = \frac{\text{luminousFlux}(a)}{\text{distance}(s,a)^2} \qquad \textbf{(L3)}$$

$$\text{ambientLightIntensity}(s) = \sum_{a \in LightEmitters} \text{directLightExposure}(s,a) \qquad \textbf{(L4)}$$

(L1) vérifie si les deux composants en question sont dans la même zone. En effet un capteur est sensible à la lumière d'une source de lumière qui est dans la même zone (dans la même pièce d'une maison par exemple), et ne détecte pas la lumière d'une source dans une zone différente. La fonction **(L1)** permet d'alléger les calculs en ignorant la lumière venant des sources se trouvant dans des zones différentes. Le cas du passage de la lumière entre les zones est traité dans la section **1.3.5**.

(L2) utilise les coordonnées (*x*, *y*) pour calculer la distance entre deux objets s'ils sont dans la même zone.

(L3) calcule l'intensité lumineuse mesurée par un capteur exposé à une seule source de lumière. Pour faire le calcul cette fonction utilise la valeur de la distance (calculée par **(L2)**).

(L4) calcule, pour un capteur donné, la somme des contributions des différentes sources de lumière (qui résultent de différents appels à **(L3)**). **(L4)** est la fonction qui calcule la valeur théorique de la mesure du capteur.

Les applications successives de fonctions sont pour l'ensemble *2DLightLawSet* sont représentées de façon arborescence sur la **Figure 9**.

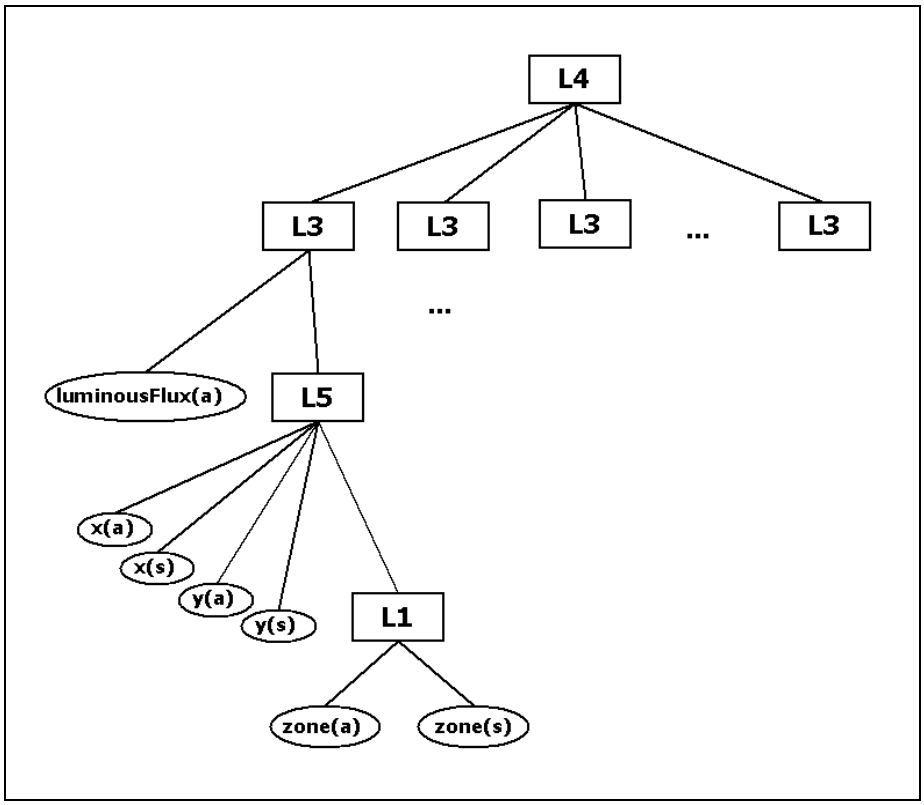

Figure 9. Arbre d'appel pour l'ensemble de lois 2D Light Law Set

1.3.3. Capteurs (Récepteurs d'effets)

Le capteur est le composant qui permet au système d'intelligence ambiante d'être informé de l'état de l'environnement et des évènements qui peuvent s'y produire. Notre approche se base sur l'utilisation des capteurs disponibles à un instant donné pour vérifier le bon déroulement des actions dans l'environnement ambiant.

Dans le modèle abstrait, le capteur est un récepteur d'effet. En effet un capteur données détecte une propriété physique particulière d'un effet, comme l'illustre l'extrait de modèle abstrait de la **Figure 10**.

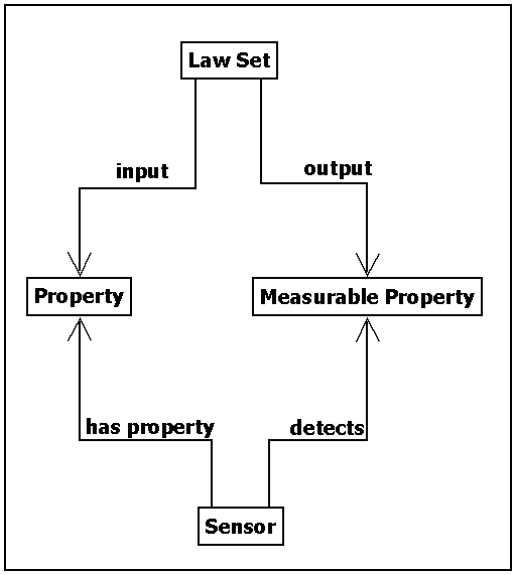

Figure 10. Entités relatives à l'objet "Sensor" dans le modèle abstrait

1.3.4. Effecteurs (Générateurs d'effet)

Dans un environnement ambiant un effecteur est l'entité qui agit physiquement sur l'environnement (voir **Figure 11**).

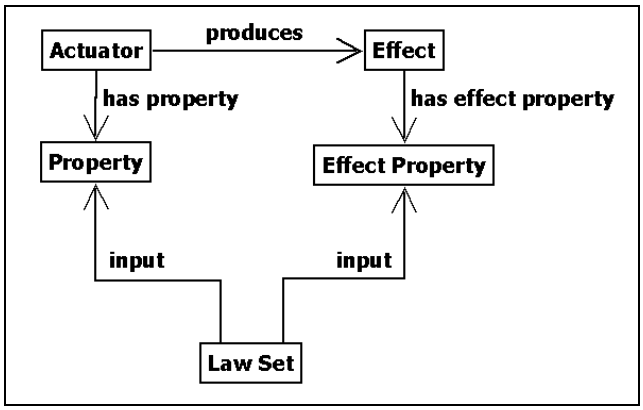

Figure 11. Entités relatives à l'objet "Actuator" dans le modèle abstrait

Un effecteur produit un ou plusieurs effets, caractérisés chacun par une ou plusieurs propriétés, qui quantifient des paramètres physiques observables. Au niveau des instances, les propriétés de l'effet et les propriétés des effecteurs (comme leur position (x, y), une valeur de tolérance, etc.) sont utilisées comme entrées dans les formules mathématiques du law-set. La valeur d'une telle propriété peut être soit statique, auquel cas la propriété garde sa valeur indéfiniment (une valeur de tolérance par exemple), ou bien elle peut être dynamique, auquel cas sa valeur change en fonction de l'état actuel de l'effecteur. Ces valeurs peuvent être déduites du modèle comportemental de l'effecteur.

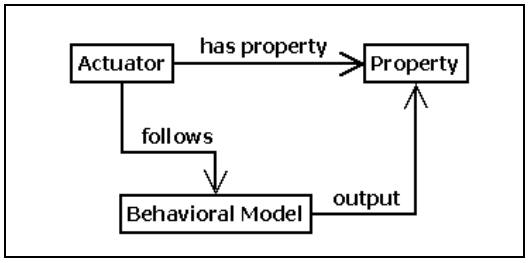

Figure 12. Modèle abstrait : représentation du modèle comportemental d'un effecteur

Sur la **Figure 12** est décrit le lien entre un effecteur (ou tout autre composant actif) et son modèle comportemental. Un modèle comportemental est une modélisation du comportement d'un composant du système. Dans cette thèse nous utilisons les automates finis comme modèles comportementaux des composants actifs. Un automate est constitué d'états et de transitions. On peut associer des actions aux transitions. Au niveau des instances, ces actions permettent de modifier les valeurs des propriétés associées aux composants actifs.

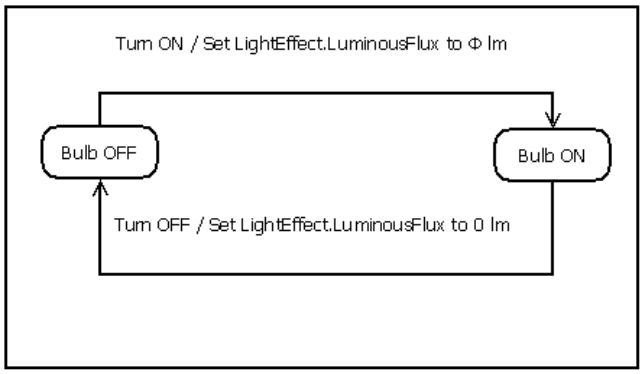

Figure 13. Exemple : modèle comportemental d'une lampe à incandescence

Sur la **Figure 13** est décrit un modèle comportemental possible pour une lampe à incandescence. On a choisi de la modéliser sous forme d'un automate à deux états (lampe allumée, lampe éteinte). La valeur du flux lumineux Φ change selon l'état de la lampe et selon la valeur définie pour le type de lampe qui sera instancié.

1.3.5. Modificateurs d'effet

Jusqu'à présent, nous n'avons traité que d'effets qui sont produits directement par un effecteur dans une zone donnée. Cependant, les effets peuvent passer d'une zone à l'autre, avec éventuellement des modifications. Par exemple, la lumière peut passer d'une pièce à l'autre à travers une porte, mais cette transmission peut n'être partielle, par exemple si la porte est partiellement fermée.

Nous introduisons donc le concept de *modificateur d'effet*, (le deuxième composant actif dans notre plateforme avec les effecteurs) qui permet de relayer un effet d'une zone à une autre. Un modificateur d'effet se comporte comme un récepteur d'effet dans une première zone, et comme générateur d'effet dans une deuxième zone. Pour calculer la valeur des propriétés de l'effet après passage d'une zone 1 à une zone 2, on commence par déterminer quelle serait la valeur perçue par un capteur dans la zone 1, puis on applique une formule de modification, par exemple simplement un taux de modification (transformation ratio) défini comme propriété intrinsèque du modificateur d'effet (voir **Figure 14**). Par exemple pour l'effet lumière :

$$\Phi = \gamma \cdot I$$

Où :

γ est le taux de modification, qui est une propriété du modificateur d'effet.

I est la valeur de l'intensité lumineuse que le modificateur d'effet reçoit dans la zone 1.

Φ est le flux lumineux que le modificateur d'effet « génère » dans la zone 2.

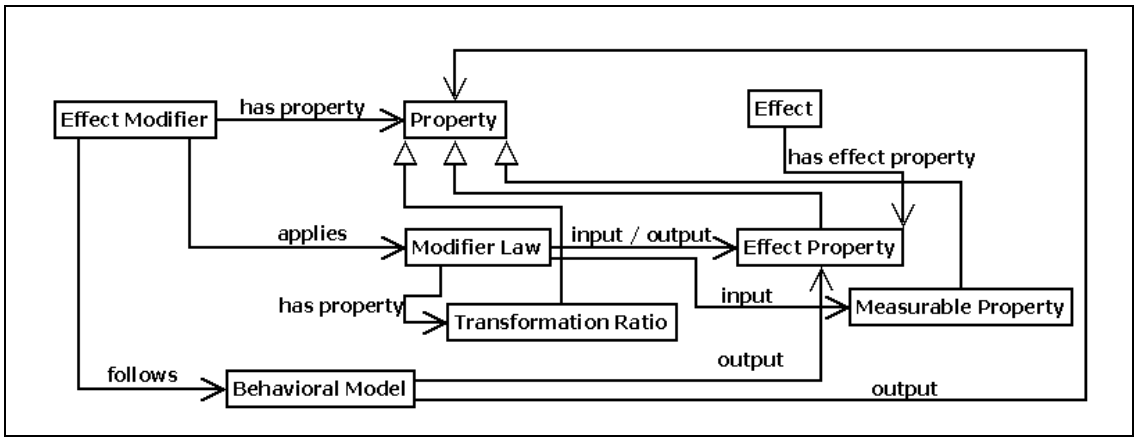

Figure 14. Modèle abstrait : modificateur d'effet

Comme pour un effecteur, un modificateur d'effet peut disposer d'un modèle comportemental.

1.3.6. Modèle de prédiction

Le modèle de prédiction contient :

- Les instances des composants réels et les valeurs de leurs propriétés.
- Les instances des modèles comportementaux des différents composants actifs, définissant à chaque instant les valeurs des propriétés des composants. Par exemple si on a une instance (bulb_1) de lampe à incandescence émettant un flux lumineux de 600 lm, Φ est remplacé par la valeur 600 dans le modèle comportemental de l'instance bulb_1, tel que représenté sur la **Figure 15**.
- Les formules de calcul, instanciées sur la situation concrète de l'environnement, déduites des law-sets. Lors de l'instanciation, on remplace les variables libres par leur valeur, et les grands opérateurs (comme par exemple ∑ dans L4) sont itérés comme nécessaire. Par exemple dans le cas de l'effet lumière, si on avait un capteur (sensor_1) et deux lampes (bulb_1 et bulb_2), les formules seraient instanciées, et on obtiendrait une formule concrète dont une représentation arborescente est donnée sur la **Figure 16**.

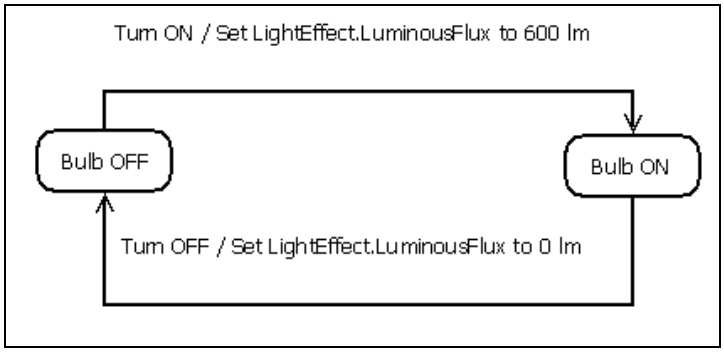

Figure 15. Automate fini de bulb_1

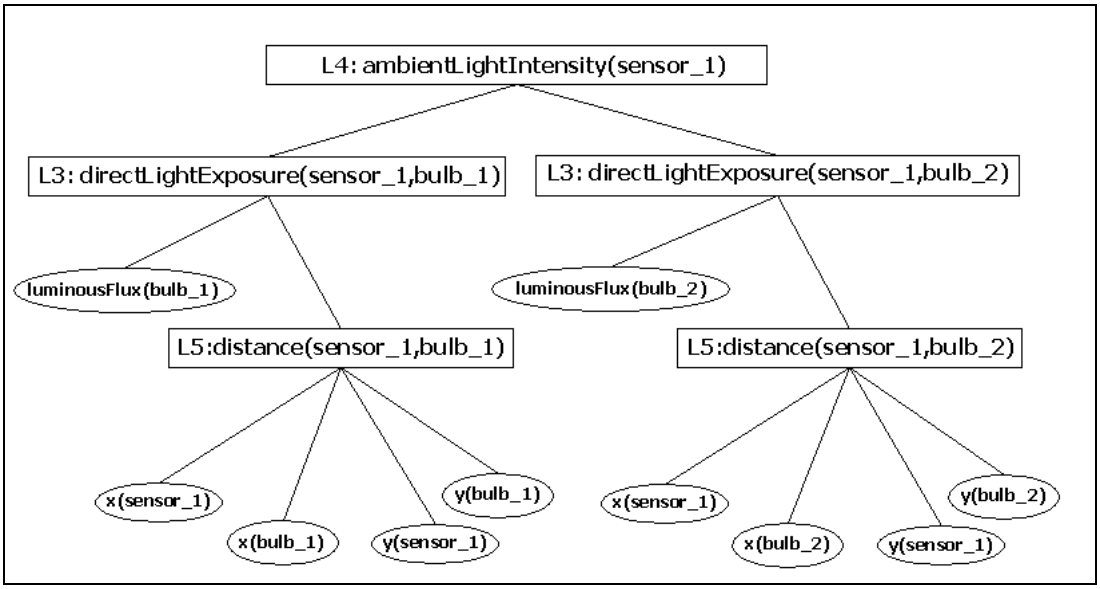

Figure 16. Formule de prédiction instanciée sur l'exemple

avec :

x(sensor_1) : le coordonnée x de sensor_1

y(sensor_1) : le coordonnée y de sensor_1

x(bulb_1) : le coordonnée x de bulb_1

y(bulb_1) : le coordonnée y de bulb_1

luminousFlux(bulb_1) : la valeur du flux lumineux produit par bulb_1

x(bulb_2) : le coordonnée x de bulb_2

y(bulb_2) : le coordonnée y de bulb2

luminousFlux(bulb_2) : la valeur du flux lumineux produit par bulb_2

1.4. Implémentation

Dans cette partie nous parlons de l'intégration de notre approche dans le projet européen CBDP **[187]**, puis nous parlons de l'implémentation de notre approche utilisant la plateforme d'exécution de modèle hétérogène ModHel'X.

1.4.1. Le projet « Context Based Digital Personality » (CBDP)

La plateforme CBDP est construite autour d'une ontologie, utilisant des composants logiciels écrits en Java et déployés sur une plateforme OSGi (Open Services Gateway initiative Framework) **[188]**.

La **Figure 17** montre une partie de l'ontologie utilisée dans le projet CBDP (en gris), à laquelle nous avons ajouté les concepts abstraits définis pour la détection de défaillance. Principalement nous avons intégré les concepts d'effet, propriété d'effet et les formules mathématiques.

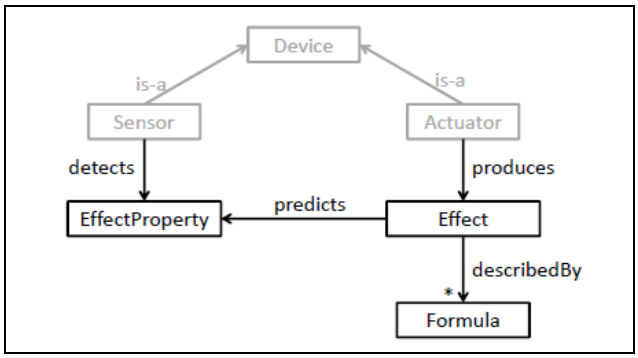

Figure 17. Intégration de l'approche de détection de pannes dans CBDP

Selon le type d'effecteur défini dans le modèle concret, le type approprié d'effet physique lui sera lié (relation « produces »), et selon le type de capteur, la propriété physique appropriée lui sera associée (relation « detects »). Par exemple pour le cas de l'effet lumière, ce modèle abstrait se concrétise comme indiqué sur la **Figure 18**.

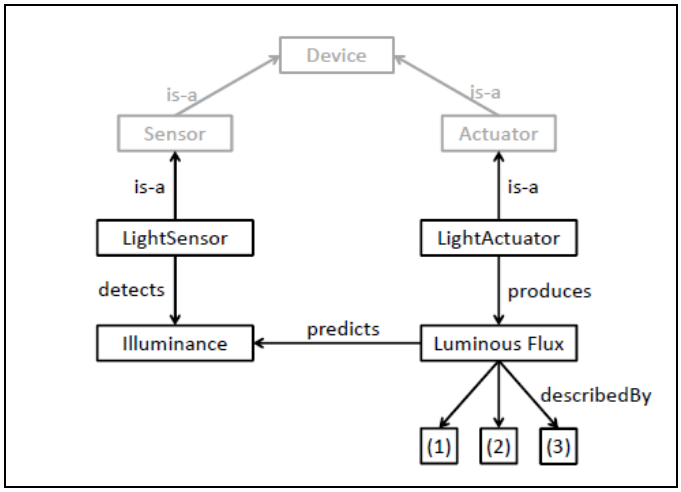

Figure 18. Le modèle concret CBDP dans le contexte de lumière

1.4.2. ModHel'X : outil de modélisation hétérogène

Comme expliqué dans la section **1.3.6**, le modèle de prédiction est un modèle hétérogène puisqu'il contient, en plus des instances des objets réels, des modèles comportementaux (automates) et des formules mathématiques (flots de données). Afin de réaliser une prédiction, il est donc nécessaire de disposer d'un outil d'exécution de modèles hétérogènes. Il faut notamment pouvoir spécifier clairement ce qui se produit à l'interface entre un modèle de type automate et un modèle de type flot de données. Nous avons décidé, au lieu de créer un moteur d'exécution dédié, d'utiliser un outil de conception et d'exécution des modèles hétérogène développé à Supélec, ModHel'X **[196]**.

ModHel'X permet la création de modèles via une API, et offre une animation graphique pour visualiser les modèles. Il permet d'exécuter des modèles en temps simulé ou en temps réel. Par conséquent ModHel'X peut être utilisé à la fois pour effectuer des simulations ou exécuter des systèmes réels. Par conséquent, le cœur du travail d'implémentation d'AmILoop consistait à construire le modèle de prédiction en utilisant l'API fournie par ModHel'X, après quoi ModHel'X peut exécuter ce modèle de manière autonome.

1.5. Exemples de détection de pannes dans un environnement ambiant : système luminaire

Dans ce résumé de thèse nous présentons les résultats du **Scenario_1** décrivant la détection de pannes des luminaires d'un environnement ambiant. Plus de détails sur ce scenario, de même que le reste des scénarios, sont fournis dans le manuscrit final de la thèse.

1.5.1. Description de l'environnement

L'environnement étudié est une pièce carrée de 16 mètres carré, équipée de 2 lampes à incandescence, une lampe fluocompacte (CFL), et deux capteurs de lumière.

La lampe fluocompacte requiert une phase de préchauffage (de 30 secondes) avant d'émettre la lumière à son intensité maximum. Son comportement est décrit dans une machine à états finis (voir **Figure 19**).

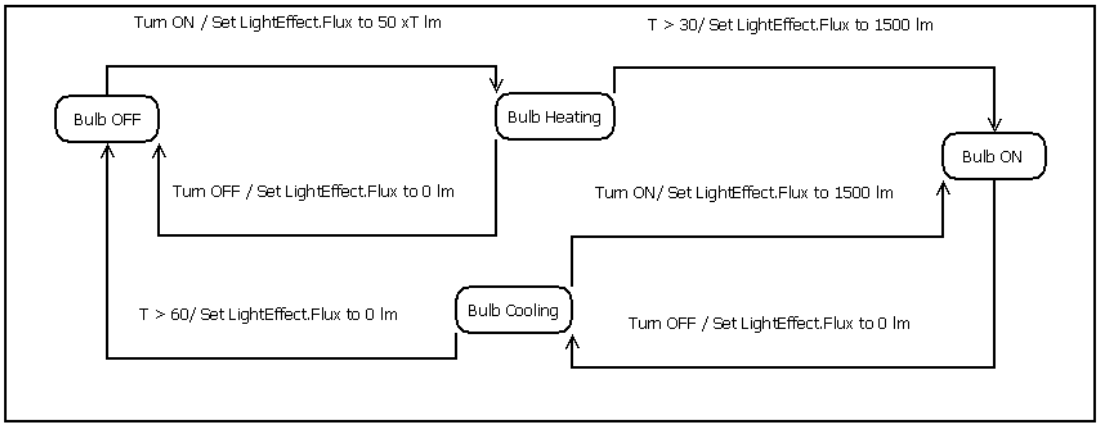

Figure 19. Machine à états finis d'une lampe fluocompacte émettant 1500lm

1.5.2. Construction des modèles

1.5.2.1. **Modèle concret**

Comme décrit dans la section **1.3.1**, un modèle concret est déduit du modèle abstrait. Il contient les différents types de composants, les liens entre les différentes classes de composants, et les phénomènes physiques que le concepteur du système juge nécessaires dans le contexte de l'application visée.

Le modèle concret du **Scenario_1** est illustré dans la **Figure 20** ci-dessous :

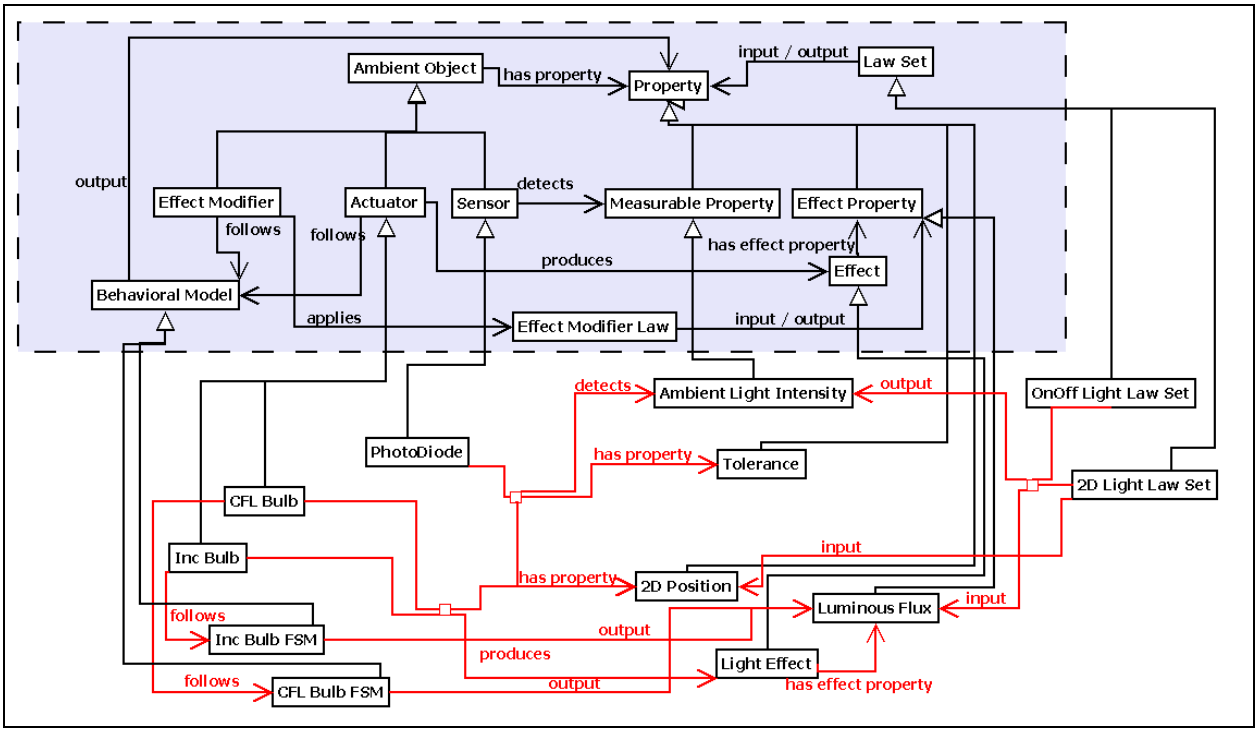

Figure 20. Modèle concret du scenario 1

1.5.2.2. **Exécution**

Le modèle concret précèdent (avec tous les objets et autres modèles qu'il contient : formules, modèles comportementaux) permet de générer un modèle de prédiction. Ce dernier est créé automatiquement dans ModHel'X en utilisant l'API correspondante.

Sur la **Figure 21** nous pouvons voir le modèle de prédiction complet (les lampes sont appelées **la1**, **la2**, et **la3**. Les deux capteurs de lumières sont appelés **ls1** et **ls2**). Le modèle utilise les données d'un service de gestion d'objet (ou moteur de contexte) afin de mettre à jour les données nécessaires pour les calculs. Ces données peuvent être des valeurs simples, utilisé directement dans des calculs, ou des événements qui déclenchent des changements d'état de certaines des automates à états finis, générant à leur tour de nouvelles valeurs alimentant les calculs.

Sur le modèle de prédiction on note les différentes parties suivantes (de haut vers le bas):

- Les entrées du modèle de prédiction venant du service de gestion d'objets.
- Les automates à états finis correspondants aux trois effecteurs lampes.
- Le modèle calculatoire (voir détails de ce modèle dans **[216]**).
- La sortie du modèle correspondant aux mesures prédites pour les deux capteurs.

La plateforme ModHel'X permet d'exécuter le modèle de prédiction calculant ainsi les valeurs qui devraient être observées au niveau des capteurs.

Validation des modèles :

Pour valider le modèle de prédiction nous avons déroulé un scénario, dans lequel les résultats de l'exécution du modèle de prédiction sur ModHel'X ont été comparés avec les prédictions déterminées manuellement.

Les validations des modèles pour chaque scénario sont détaillées dans le manuscrit de thèse.

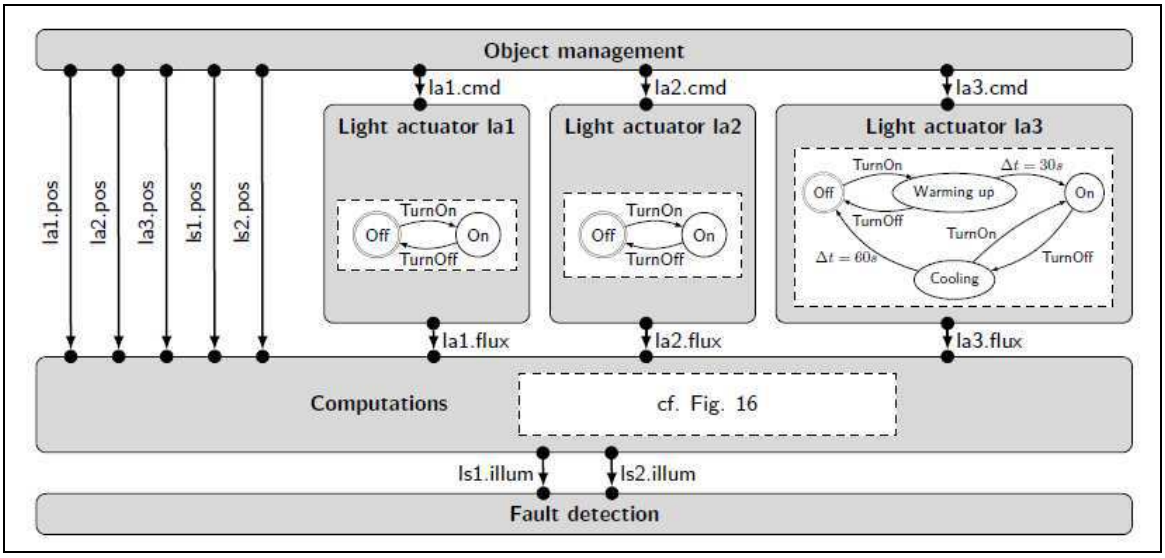

Figure 21. Modèle de prédiction exécuté par ModHel'X

1.6. Conclusions et perspectives

1.6.1. Notre Contribution

Dans cette thèse nous avons présenté une nouvelle approche pour effectuer les tâches de détection et diagnostic de pannes dans les environnements ambiants. L'approche proposée est fondée sur une modélisation des phénomènes physiques que nous avons appelée **effet**. L'effet est le seul lien entre capteur et effecteur, ce qui permet de découpler les entités au moment de la conception du système. Les liens concrets sont déduits automatiquement au moment de l'exécution, en fonction des effecteurs et capteurs réellement présents.

A partir d'un examen de ces liens, nous générons un modèle de prédiction qui calcule les valeurs attendues au niveau des capteurs. La comparaison entre les valeurs théoriques calculées et les valeurs réelles permet de détecter les défaillances.

Nous avons implémenté notre approche sous la forme d'une plateforme AmILoop. Les modèles de prédiction, qui sont de nature hétérogène car ils contiennent à la fois des flots de données et des comportements, sont exécutés à l'aide de l'outil ModHel'X.

1.6.2. Perspectives

1.6.2.1. **Modificateur d'effet avancé**

Les modificateur d'effet actuellement implémentés supposent que le taux de transformation d'effet est le même dans les deux sens, sauf que dans la réalité ce n'est pas toujours le cas. Nous pouvons alors imaginer une modélisation plus poussée des modificateurs permettant de prendre en compte les zones source(s) et destination(s) pour appliquer la fonction appropriée.

1.6.2.2. **Diagnostic de pannes**

Dans notre approche nous n'avons pas pu implémenter une technique de diagnostic de pannes. Nous avons proposé une architecture générique pour le diagnostic qui se base sur l'utilisation d'un moteur de diagnostic qui exploite un modèle de diagnostic approprié. Plusieurs techniques concrètes peuvent être utilisées pour le diagnostic.

Modèle probabiliste. Nous pouvons envisager l'utilisation de modèles probabiliste pour décider lesquels des composants concernés par la panne (lesquels peuvent être déduits à partir du modèle de prédiction) sont plus susceptibles d'être la cause de la panne détectée. L'idée est de calculer pour chaque composant une probabilité de défaillance.

Arbres de décision. Nous pouvons aussi imaginer un modèle de diagnostic qui se base sur une arbre de décision pour répondre à une série de questions permettant de réduire la liste des objets qui peuvent causer la panne détectée. Par exemple un tel arbre peut mener à vérifier s'il y a d'autres capteurs qui détectent la même anomalie, de façon à confirmer ou pas la défaillance de l'effecteur.

Ontologies. Nous pouvons aussi appliquer un raisonnement plus sémantique sur les informations déduites de la phase de détection de pannes. Dans un tel cas une ontologie **[189]** peut être utilisée pour décrire (sémantiquement) l'environnement, ses composants et les liens entre les entités impliquées. On peut alors spécifier des règles de raisonnement qui permettent de raisonner sur les informations de l'ontologie. Ces règles sont exploitées par un moteur d'inférence qui peut, en plus, tirer parti des valeurs et données déduites par le modèle de prédiction.

1.6.2.3. **Diagnostic portant sur les tâches des utilisateurs**

Notre plateforme de diagnostic permet de détecter des pannes du système, mais elle ne prend pas en compte les actions des utilisateurs. Au contraire des actions du système, les actions des utilisateurs sont plus imprévisibles, et par conséquent plus difficiles à diagnostiquer. Des travaux sont menés dans le domaine de la modélisation des tâches utilisateurs en tenant compte des particularités des environnements ambiants **[211]**.

1.6.3. Conclusion

Les systèmes ambiants ont un potentiel de développement important à l'avenir. Ils seront probablement utilisés par des personnes pour leurs activités quotidiennes, sans même qu'elles s'en aperçoivent. La fiabilité et la capacité d'auto-diagnostic de ces systèmes seront donc des propriétés très importantes, voire critiques.

Notre travail porte sur l'auto-diagnostic. Nous proposons une solution originale pour la détection de pannes dans le contexte ambiant. Cette solution pourra à l'avenir être complétée pour prendre en compte le diagnostic des pannes.

Chapter 2:

Introduction

Chapter 2.

Introduction

2.1. Context & Motivation

Our works are in the domain of Ambient Intelligence (AmI). Ambient intelligent systems are interactive systems composed of many heterogeneous components that can be mainly divided into sensors, using which the system observes its surroundings, and actuators, through which the system acts upon its surroundings in order to execute specific tasks. The aim of the tasks is to provide comfort to the occupant of the environment, improve life in general, supervise and assist users that need help or care. Ambient Intelligent techniques are used in many contexts such as homes, hospitals, factories, etc. in order to add a layer of intelligence to the environment, which improves the overall experience of the users (when it is for personal use) and develops the productivity (when it is used in a professional and/or industrial setting) according to the context of use.

There are many aspects of the Ambient Intelligence field that are subject to research and development works, such as the hardware and technologies that are used in a smart environment, the concepts and techniques that add an intelligence layer to Ambient Intelligent environment, contextualization and self-configuring techniques, task modeling in Ambient Intelligent environments, etc. There are three main types of tasks performed in an Ambient Intelligent environment: tasks performed by the users (very difficult to predict), tasks performed by the system (via the actuators) and tasks executed as part of user-system interactions. In the field of Ambient Intelligence, these tasks are approached and studied according to different angles (see **Figure 22**).

Our work concerns tasks that are performed by the system via the actuators. We aim to supervise the actions of the system in order to detect any potential malfunction. In addition to the fact that actuators are very prone to faults, the heterogeneity of Ambient Intelligent systems' devices makes detecting the real cause of the fault a difficult mission. Experts in the domain of Ambient Intelligence (AmI) are interested in designing adaptive, self-healing, and fault tolerant ambient intelligent systems. We reckon that such systems should be able to autonomously "discover", "understand" and "correct" the faults that occur in the execution of system tasks in an ambient environment. Discovering faults is called *fault detection*, and understanding faults (deducing more information about it, including the source of the fault ideally) is called *fault diagnosis* (see **3.3.1** for detailed definition).

Our work's overall goal is to create such a fault tolerant system. In this thesis we propose a high level and scalable architecture for a fault detection and diagnosis layer for Ambient

Intelligent Systems. We detail and validate, via examples, the fault detection part. We explain how the general architecture allows the use of most state of the art techniques to diagnose detected faults.

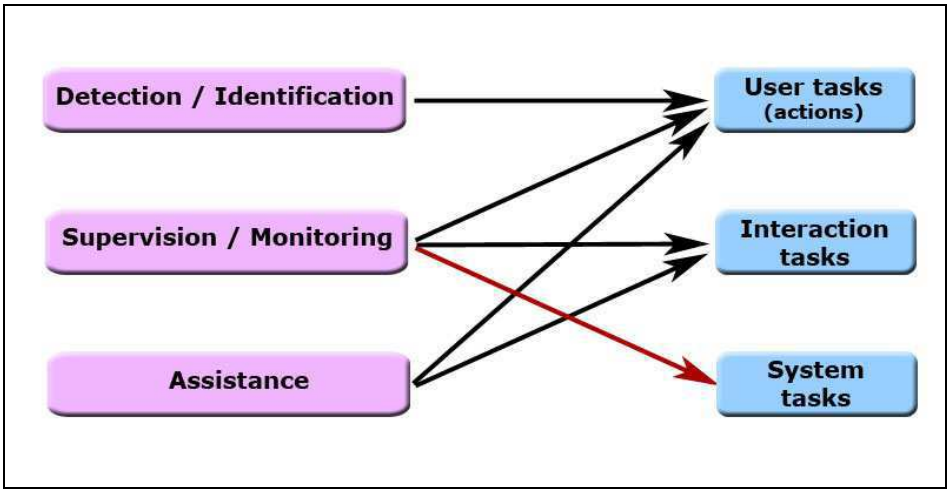

Figure 22. Main research questions studying tasks in an AmI environment

The field of Ambient Intelligence refers to interactive systems in which the processing and interaction capabilities are embedded into everyday objects, thus facilitating the installation of an intelligence layer creating a smart environment. Smart homes, hospitals, public transport, and factories, are some examples of application of smart environments. Applications range from enhancing everyday life tasks to monitoring and guaranteeing patients' safety in hospitals. The main objective of an Ambient Intelligent environment is to address the needs and preferences of the user. To do so, the ambient system interacts with its surroundings; it perceives the state of its surrounding using sensors, and acts accordingly upon the environment using actuators. On the one hand, to ensure the achievement of its goals, the ambient system depends strongly upon the proper conduct of the tasks that are performed by its actuators. On the other hand, Ambient Intelligent systems are designed to keep a certain level of non-intrusiveness in order to avoid any discomfort of users. That is why tasks are usually executed in the background in a way that is unnoticeable by the user. Such a requirement makes it unacceptable to flood the user with a large number of fault detection data. Conversely, not informing the users of detected faults may cause that users continue to rely on failed services without noticing **[1]**. This can even be dangerous in some cases, for instance in the case where an Ambient Intelligent system is used for monitoring patients in a hospital, while some patients' data are corrupt due to a detected, but undeclared (ex. until next maintenance intervention), actuator or sensor failure. This characteristic, which is not trivial to attain, constitutes, among others explained later, one of the particularities of Ambient Intelligent systems that motivates research in this field.

In this context we want to endow such systems with tools allowing them to check autonomously whether or not systems tasks are performed properly. As a matter of fact, when an ambient system sends out orders to an actuator, the proper way to verify whether an order has been executed properly is to exploit the sensors' readings in order to ensure that the state of the environment has changed as expected. For instance, when the system activates a light bulb, the hardware infrastructure and communication capabilities allow the system to verify whether the order has been transmitted properly and that the electric circuit of the light bulb has been closed. However, many factors could stop the light from being emitted, for instance the light bulb could have been damaged and so it would not be lit properly, or the bulb could have been covered by a non transparent object, etc. So to verify that the light has really been switched on, the readings of the proper light sensors must be considered (see **Figure 23**). New challenges arise: first selecting

relevant sensors (here light sensors), second selecting only sensors that are exposed to the actions of specific actuators (here light sensors exposed to, and affected by, the light emitted by a given light bulb). A solution to that from control theory consists in pre-determining closed control loops using ad-hoc sensors. However, one of the main particularities of ambient systems is that, unlike traditional systems, physical resources (mainly sensors and actuators) are not necessarily known at design time. In fact they are dynamically discovered and may appear and/or disappear at run-time, so the solution using pre-determined control loops cannot be adopted in such an open environment.

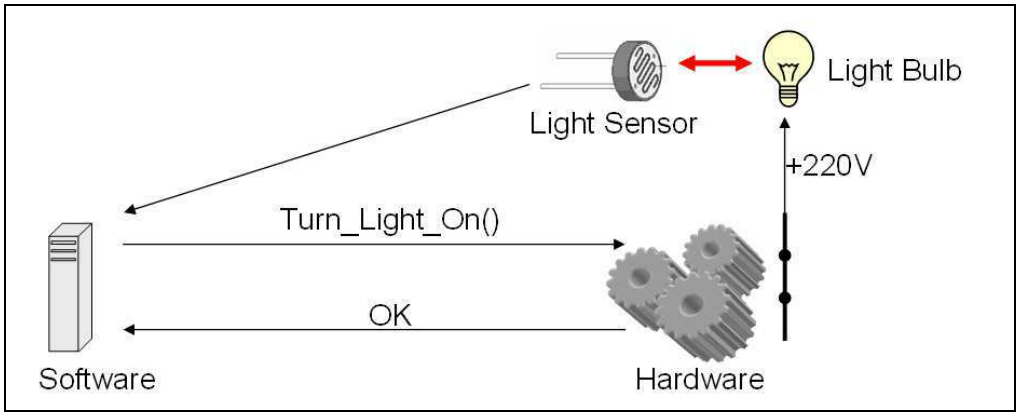

Figure 23. Classic ad-hoc control loop for system diagnosis

With this work we propose a solution that allows the automatic and dynamic construction of the proper links between actuators and sensors in ambient systems by exploiting available resources at a given time, and using them to perform fault detection and diagnosis at run-time. The approach is based on the modeling of the physical phenomena (that we call effects) expected to occur in the environment when a given actuator is activated. Effects are characterized by physical laws that can be modeled at various levels of details. These laws depend on physical parameters that are associated with actuators and sensors types. By exploiting modeled information and physical laws at any given time, the system is able to automatically create associations between actuators and sensors at run-time. Then by performing the proper calculations, the system deduces the measurements expected from a given sensor when a certain action is performed by an actuator (for instance, an increased temperature level may be expected within a certain time lapse when a heating system is activated). This way, the system is able, first by comparing these calculated values with the actual sensors readings, to detect the existence of faults (fault detection), then by reasoning over the available information (from the diagnosis model), to produce an accurate diagnosis (finding the fault source, which could be the actuators, the sensors, or any other modeled system object) at run-time without requiring the explicit coupling of actuators and sensors at design time. In fact, the relations between the actual components are entirely deduced at run-time from the characteristics of actuator and sensor types (physical phenomena produced by an actuator type and detectable by a sensor type), and from characteristics of actuators and sensors that are instances of these types (actual area affected by actions of an actuator, actual area visible by a sensor, the positions of actuators and sensors and the distance between them, etc.). Therefore it is well adapted to the **openness** of Ambient Intelligent Systems. Moreover, Ambient Intelligent Systems are very dynamic in the sense that the states (on/off state, output value, position, etc.) of the actual components are constantly changing, hence deducing faults in such a **dynamic** setting might depend on the previous state of the system (overall state or some components states) and of the environment. For instance, an error consisting in an unexpected drop in light level is detected by comparing the current light level with the previous one. In other cases, a progressive increase in temperature level may be expected within a certain time lapse when a heating system is activated, in which case the current

temperature value at every point in time is calculated by adding the calculated increase (or decrease) of temperature relative to a previous temperature state. Thus, it is crucial to consider their overall temporal behavior. For this reason, we also introduce temporal extensions to the Fault Detection and Diagnosis framework.

2.2. Outline of the thesis

This thesis is organized as follows. *Chapter III* is a state of the art, in which we introduce the field of Ambient Intelligence, and we define some related fields and concepts and show some similarities and differences between them. In the same chapter we identify some of the works that have been done in the field of fault detection and diagnosis in the field of automatic control, and some of the adaptations of these methods and some new methods of fault detection and diagnosis in the field of ambient intelligence and pervasive computing. We focus on the limitations of these approaches; especially the non compatibility of these methods with some of the main particularities of ambient intelligence field (mainly its openness and dynamicity). These particularities are also highlighted in this chapter. *Chapter IV* details our Fault Detection and Diagnosis Framework. We describe our framework's architecture, composing models, these models structure, hierarchy, nature, and (in some cases) the sub-models within the models. We also explain how these models allow us to overcome the challenges faced by working in an Ambient Intelligent environment and ensure the good execution of the fault detection and diagnosis tasks. The latter tasks are also detailed in this section. In *Chapter V*, we focus on the implementation aspect of our framework. More precisely we detail the integration of our fault detection and diagnosis framework into an Ambient Assisted Living application, and the development of a java simulator in order to test and validate (by simulating scenarios) our Fault Detection and Diagnosis approach. In *Chapter VI* we use our simulator in order to test the framework in more real life scenarios. The scenarios presented in this section are used for evaluating the Fault Detection and Diagnosis tasks accuracy. We validate our approach by simulating some scenarios via ModHel'X, which is a framework for heterogeneous modeling and for simulating the execution of multi-formalism models. *Chapter VII* is a conclusion where we resume the work in our thesis and where we also point out some of the limitations; then we discuss some of the possible future works that would help overcome the mentioned limitations and ameliorate the our approach's performances and the accuracy of the Fault Detection and Diagnosis results.

Chapter 3:

State of the Art

Chapter 3.

State of the Art

Since humans started building machines and relying on them to perform tasks, ensuring the proper functioning of these machines had become an important matter. As machines evolved into more complex systems and their uses broadened into almost every field, all humans became, directly or indirectly, dependent on the proper operation of these systems. Nowadays many fields use complex systems, among which we cite: agriculture, communication and information, transportation, healthcare, manufacturing etc. As human safety and lives became reliant on these systems, avoiding malfunctions became more and more of an issue, and fault detection and diagnosis became an essential research domain. At first, many studies have been made for diagnosing systems in the field of automatic control. Later systems evolved into more complex and digitalized systems, which led to the appearance of modern control systems **[2]** that are able to meet increased performance and safety requirements of modern complex systems. The general idea behind control systems is to create models representing the diagnosed system, and draw fault detection and diagnosis conclusions from the models or from superimposing the models and the real system. These studies have been the foundation for later fault detection and diagnosis techniques for various other domains with different types of systems. Among these systems, we focus in this thesis on modern pervasive, ubiquitous computer based ambient intelligent systems.

In the state of the art we will first define ambient intelligence, ambient intelligent environments, pervasive and ubiquitous computing systems; exact definitions are to be set, relations between concepts, and the particularities of each of them are to be detailed. We also show the importance of having fault free systems in the ambient intelligent systems context. In the second part we make an overview of fault detection and diagnosis techniques in the field of automatic control, from which we will fix our terminologies and definitions that will be adopted for the rest of the dissertation. We conclude by showing how fault detection and diagnosis techniques from the field of automatic control ought to be inspired from and/or adapted for the new complex, dynamic and heterogeneous Ambient Intelligent systems. We also mention some of the particular challenges in the field of Ambient Intelligence that should be overcome in order to efficiently detect faults and perform diagnosis and consequently obtain a fault free Ambient Intelligent system. The third part discusses some existing works that have been done in the ambient intelligence domain for fault detection, diagnosis and self-healing in order to provide fault-free systems.

3.1. Ambient intelligence (AmI)

3.1.1. From Artificial Intelligence to Ambient Intelligence

Ambient Intelligence (AmI) is a vision of the future world, in which technology is present everywhere but invisible. In such context users are not necessarily aware of using the technology embedded in their surroundings. The technology in an Ambient Intelligent context is available at all times with simple interactions with any device. Such technology ought to be smartly adaptive to different contexts of use and to different users' preferences. The technology is also

autonomous in the sense that it is self configuring and self healing **[3]**. In **[4]**, Ambient Intelligence is considered as the next step in the evolution of Artificial Intelligence. In fact the authors show, while citing previous works describing the deep bond between the two fields **[5][6]**, that Ambient Intelligence cannot be achieved without Artificial Intelligence's concepts, methods, techniques and technologies. From **Figure 24 [4],** Ambient Intelligence's evolution from Artificial Intelligence is depicted via the parallel evolution of three layers:

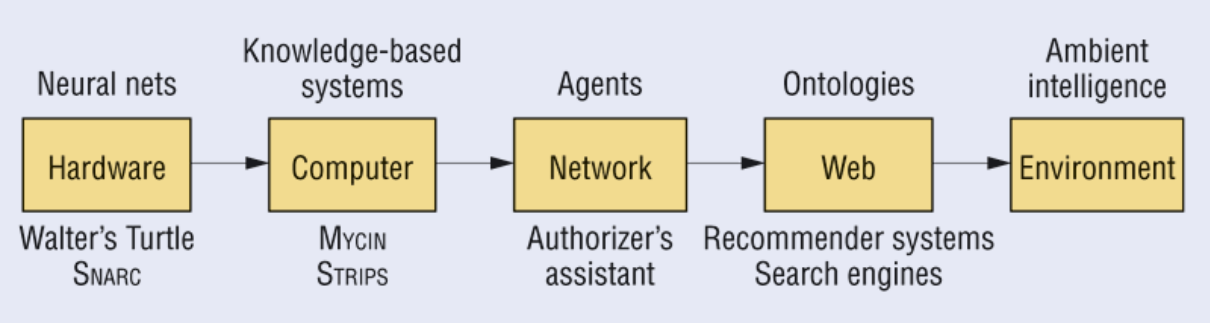

Figure 24. Ambient intelligence as an evolution of artificial intelligence

(a) *The hardware (also called the operational layer)*, which evolved from analog devices and early computers, to modern digital computers. These computers were then connected in networks. As networks became more democratized and easy to access the web emerged. Then as heterogeneous modern digital devices got connected to the web it is our whole environment that became the hardware layer of our technologies. This was made possible by the miniaturization of electronics and the fact that small devices with strong (compared with earlier technology) computational capacities can now be bought at affordable prices.

(b) *The formal AI concepts, tools and techniques*, which are used to properly exploit the hardware layer in order to generate intelligence. Making the best of the hardware advances, these concepts evolved. So from artificial neural networks, which were implemented on the available analog hardware devices in order to solve particular tasks (such as in **[7]** and **[8]**), tools complexified into knowledge-based systems that are able to make intelligent decisions about a specific domain. As hardware evolved into networks the concept of distributed agents was created and the decision making tool was decentralized. Ontology-based techniques evolved as the networks became the web and the quantity of information exploded and it had to be more intelligently exploited. When the information available in the web became accessible and embedded into almost all modern devices, the tools, recently, evolved into what we now call Ambient Intelligence.

(c) *The applications that resulted from applying AI techniques to the hardware layer,* such as, William Grey Walter's educational robots (called Walter's turtles), which were designed in the late 1940's and were used in computer science and mechanical engineering training. In the 1980s they were equipped with pen mechanisms allowing the use of Logo programming language to create designs on papers for educational purposes **[9]**. Also using analog hardware, the SNARC (Stochastic Neural Analog Reinforcement Computer) was a self learning calculator created by Marvin Minsky and Dean Edmonds. SNARC was a neural network based machine implemented using vacuum tubes as a hardware platform. As better performing digital computers were used with knowledge based systems simple inference engines were made possible and expert systems such as MYCIN **[10]** emerged. MYCIN was an expert system that recognized infections, identified bacteria that caused them, and recommended the proper antibiotics. Other expert systems were also developed such as STRIPS (Stanford Research Institute Problem Solver) **[11]**, which is an automated planner that permits the execution of a sequence of actions in order to achieve a specific goal. Then computers got connected in networks and agents were used to

create collaborative applications such as Authorizer's assistant **[12]**, which is a rule-based system used, on an banking IBM mainframe networked environment, by American Express to analyze credit cards requests and complement online credit authorizations. With the development of the web recently and the fact that the amount of data became very important and very various on the web, search engines are continuously updating their techniques using new very efficient (faster and more precise) search algorithms. New concepts also emerged such as ontologies that added a semantic layer to the information available on the web. This makes search requests in a more human like form and results more 'intelligent'. A known application that uses this is the recommender systems. Recommender systems predict and recommend data that is interesting to the user according to previous activities and preferences **[13]**. Now we have ambient intelligent applications that exploit all the communication and computation capabilities that are incorporated in most modern devices that surround us. The remarkable progress in Ambient Intelligent technology has created a variety of new intelligent systems that we discuss furthermore as we list a number of smart environments in **3.1.4**.

According to the above definitions, (b) and (c) can be called the intelligent layer that envelops the operational layer (a) in an Ambient Intelligent system.

The makers of these new technologies are constantly trying to make them more user friendly, by making them easier to use and more intuitive. This has resulted in the spread of this technology worldwide among all types of users; add to that the miniaturization of the hardware, the result is the embedding of the computational and communication capabilities into almost everyday objects. This is called ubiquity **[14]**. Ubiquitous technology (also called embedded technology) has allowed these objects, besides allowing technology to be everywhere, to act in a very unnoticeable manner by the users. This has made it possible for technology to disappear in the background and task that they help accomplish became naturally part of everyday life activities. The embedding of technology in the environment is boosted by the fact that new systems are equipped with more intuitive multi-modal interfaces **[15]**, which allow users to interact with their surroundings in a natural way. This idea is pushed even further by the use of spoken language to receive commands from the system's users **[16]**. In **[17]** the authors proposed an approach in which speech and gesture are used simultaneously as a mean of interaction.

The field of Ambient Intelligence is recent and technologies and terms that are used are evolving as researchers publish new works. That is why, in the next section, we define some of the most common concepts and technologies that are related to the field of Ambient Intelligence, and we show some similarities and overlapping characteristics between them. Another factor that contributes to varying definitions of Ambient Intelligence (and of some concepts when used in Ambient Intelligence context) is due to the fact that it can be found (applied) in many contexts. So given the diversity of potential applications, the definitions naturally extends to other areas of science like education, health and social care, entertainment, transportation, etc. Ambient Intelligence started to be used as a term to describe this type of developments about a decade ago **[18]** and it has now been adopted as a term to refer to a multidisciplinary area which covers a variety of other fields of computer science as well as engineering **[19]**, as illustrated in **Figure 25 [20]**. Some concepts and definitions from these fields might overlap with Ambient Intelligence but they sill are specific fields in comparison with the field of Ambient Intelligence. For instance human-computer interaction (HCI) is a very important part of Ambient Intelligence. As a matter of fact observing users' needs and requests in order to satisfy their requests and preferences is a central part of Ambient Intelligent systems. However the field of HCI does not cover Ambient Intelligence. The same thing can be said about Pervasive Computing (see next paragraph), Multi-Agent Systems **[21]**, Sensor Networks and Robotics.

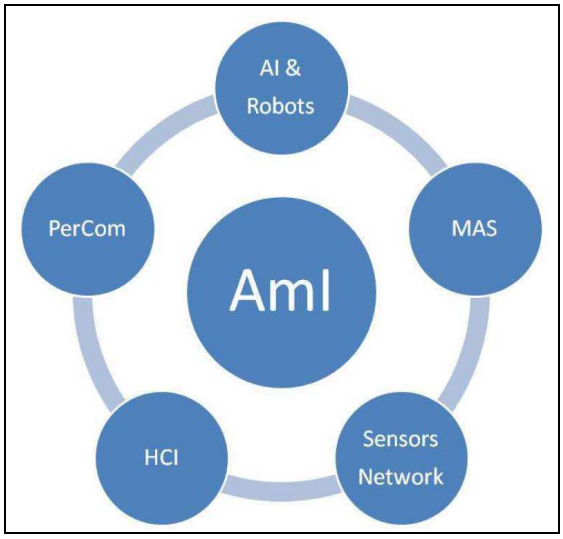

Figure 25. AmI and several scientific areas

Another recent technology, which is not a direct evolution from Artificial Intelligence, worth mentioning in this context, is sensor networks. As more and more modern systems and techniques rely on sensor networks, the latter became a field by itself. See **3.2.4** for further details and positioning of our work relatively to the field of sensor networks.

3.1.2. Definitions

"The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it."- M. Weiser **[22]**

In this section we define, from Ambient Intelligence perspective, concepts and technologies that are related to the field Ambient Intelligence.

In recent years, the field of computer science has gone though fast and major progress. We are already living in a connected world on a human level. With the embedding of intelligence and computational capacity into the devices in our environment we are going toward the social network of objects. This will further improve the way we interact with our environment. This embedding of intelligence everywhere all the time and the technical possibilities generated from such technology are subjects of studies that are being explored in an area called Ambient Intelligence.

Ambient Intelligence has different definitions given by researchers who looked at the concept from different angles and points of interests **[23]**. For instance in **[18]**, an ambient intelligent system is viewed as a **sensitive** and **responsive** system. It is sensitive since it recognizes the presence of users, and responsive since it acts in response to our presence and in accordance to our actions and requests. In **[24]**, which is a visionary work done in 2001 imagining different scenarios in a world of ambient intelligence; the authors adopt the same definition, however, in addition to the sensitive and responsive characteristics, the systems in the scenarios emphasized on the **transparency** of the systems' services. This transparency is made possible by the **ubiquitous** character of the technologies used in Ambient Intelligent systems; that is to act in an unnoticeable manner. Work in **[25]** shares the same vision of transparency by assuming that the use of Ambient Intelligence brings humans an easier and entertaining life while disappearing into their environment background. The work in **[26]** adds to the **non-intrusive** and transparent characteristics, the **intelligence** characteristic. Intelligence and transparency are also the main characteristics in the definition of an Ambient Intelligent environment in **[27]** alongside with

sensitivity to the current state (or context), in order to properly adapt to the needs of the user. **Adaptability** of Ambient Intelligent systems is also the center of interest in **[28]**.

Another, very important, characteristic of Ambient Intelligent systems, which they share with the field of autonomic computing, is **self-management [29]**. This characteristic is very much bond to **context awareness [30]** in the sense that it is very important for the system to be aware (via its sensors) of the current context (state of all elements involved in the environment) in order to choose the best automatic 'next action' and provide the most suitable service, and to the appropriate user(s). Self-management is a very general characteristic that can be divided into a broad list of more fine-grained characteristics, such as **self-configuration**, **self-adaptation**, **self-optimization**, **self-protection** and **self-healing**. The latter is possible after **self-diagnosis [31]**.

What we notice from these different, but overlapping, definitions is that Ambient Intelligent systems have some similarities and differences with fields such as Ubiquitous Computing, where computers (technology in general) are disappearing into the background and where transparency is one of the main characteristics. However the main difference is that the goal of Ubiquitous computing field is a technological goal, which is to successfully build portable technologies that can be embedded discreetly and efficiently into the environment.

Pervasive Computing, however, is a field that uses ubiquitous technology to benefit from the transparency characteristic **[32]**, but focuses on the accessibility of the information and the ease of use of the technology, which are user-centered goals. So Pervasive Computing can be defined as a field where digital and physical devices are seamlessly integrated together in the user's environment, and where users can access information from any device with the same facility of accessing it from computers.

Even though pervasive computing surrounds the user in its environment, it should not intrude the user's consciousness **[22]**. To do so effectively, the system should be able to adapt its behavior according to the context of use. This specific aspect of the technology is thoroughly discussed in the field of Context-Aware Computing. Sensitivity, **responsiveness**, and adaptability are the main characteristics of the field of Context-Aware Computing.

Ambient Intelligence also incorporates aspects of artificial intelligence as the tasks of analyzing, understanding and adapting the system behavior to the user needs and preferences in most times resort to using machine learning algorithms and agent-based paradigm. However analyzing, understanding and adapting to users makes Ambient Intelligence field interacts more with human senses (hearing and vision), human language, human knowledge, human reasoning, and human intelligence in general; hence one of the particularities of Ambient Intelligence field **[33]**.

Moreover Ambient Intelligence goal can be resumed in "Intelligence Everywhere", which we reckon is a more general goal than the other related field. More than that, from a human user point of view, Ambient Intelligence can be seen as the ultimate goal to be achieved by most of the mentioned modern technological and scientific research fields. That is why we can define Ambient Intelligence as a field that draws features from Pervasive Computing, Context-Aware Computing, Ubiquitous Computing, and Artificial Intelligence, augmenting sensitivity, responsiveness, adaptability, and transparency of the system, in order to enhance the users' everyday life tasks, comfort and overall experience.

3.1.3. Ambient Intelligence and human interaction

Ambient Intelligent systems are systems that are able to interact with humans in an 'intelligent' way in order to provide services. To provide these services with a humanly perceived intelligence, Ambient Intelligent systems must be context-aware. Context awareness requires adapting to users preferences and different situations.

3.1.3.1. **Context-aware human interaction**

To acquire knowledge of users' preferences, Ambient Intelligent systems can either learn them (via observing and recording users' behaviors) or having them pre-set (by the user itself or by an expert). From a technological point of view, the learning method requires the use of sensors, this means the analysis and fusion of various input data (for instance using Kalman filters **[34]**, statistical techniques **[35]**, or other techniques for combining data from different sources **[36]**). These inputs come from different sensors readings, and they may also come from human interaction. To make Ambient Intelligent systems as human friendly as possible, the means of interaction between humans and Ambient Intelligent systems are designed in such way that humans use intuitive ways to communicate with, and/or control, Ambient Intelligent systems. Therefore many techniques have been adopted from the field of human computer interaction in order to facilitate this task. Among these techniques there are various automatic recognition techniques, such as speech processing **[37]**, motion tracking and gesture recognition **[38]**, facial expression recognition **[39]** and emotion recognition **[40]**.

As showed in the previous section, Ambient Intelligent field is a multi-disciplinary field in the sense that it is multi-technological (a wide range of heterogeneous devices are being used) and multi-modal (different concepts and technologies are being used). Among the challenges that arise in a field with such heterogeneity is to provide a practical feedback to the users of the system. Feedback is very important to keep users aware of what is happening to the systems. Ambient Intelligent Systems have to use the easiest to understand, yet the richest representation possible, hence the importance of knowledge representation in Ambient Intelligent systems **[4]**. This uses many automatic translation techniques, statistical and knowledge-based approaches.

Ambient Intelligent Systems add a layer of 'intelligence' to the users' surroundings. To do so Ambient Intelligent systems are trying to get close to humans in their way of observing and analyzing their surroundings and situations. In nature, vision is one of the richest sensorial inputs. For computers vision is a geometric reasoning problem to be solved. Computer vision comprises many areas, such as image acquisition, image processing, object recognition (2D and 3D), scene analysis, and image-flow analysis. In practice computer vision can be used in different cases in the field of Ambient Intelligence field, such as identifying traffic problems, patterns, or approaching vehicles. It can also be used to identify human gestures (to control devices for instance) or human facial expressions to identify emotional states. Since Ambient Intelligent systems deals mainly with humans, social and emotional factors ought to be considered. For instance some social aspects, such as receiving visitors, or emotional aspects, such as being in a bad mood, might make a person not wanting to watch his or her regular favorite program. Some work has been done in the field of AI on affective computing **[41]** and social computing **[42]** and are used in the field of Ambient Intelligence. According to **[43]** there is a correlation between the person's emotional state and facial and body expression, which makes it possible to detect a person's emotional state based on a picture or a video using computer recognition of facial expressions techniques.

The operational layer of Ambient Intelligent system is composed of many technologies. These technologies are from a multitude of disciplines involving human-computer interfaces, networking and pervasive computing, and services architectures. Such technologies are based on actuators and sensors **[20]**. The main drive of the technology is satisfying the user preferences. As a matter of fact, many Ambient Intelligence-based smart environment applications were developed based on user-centric data extraction and decision making techniques. That is why many of these applications interact with users using gesture interpretation, detection of regions of interest (associating them with particular events or situations), and interactions between the user's behavior or mood and the surrounding environment devices **[44][45]**.

Moreover, this technological progress that have been made, especially in devices such as cameras, embedded processors, and the fact that these technologies have become cheaper and more democratized, contributed in making it easier to realize original real-time solutions such as immersive human-computer interfaces for virtual reality experiences. These solutions can be used to revolutionize various types of applications such as gaming, teleconferencing, smart presentations, and gesture-based control, and even patient monitoring and assistive technologies **[20]**.

A very remarkable aspect of these new Ambient Intelligent technologies is the fact that on an operational level (hardware) they rely on very heterogeneous devices. They are in fact heterogeneous in their technology, purpose and means of communication. This led to the rise of new challenges in communication (incompatibility of communication protocols), synchronization, fault diagnosis, performance, etc. In fact Ambient Intelligence is a field that benefits from works in other fields in order to overcome such challenges. Among these techniques there are data fusion, event interpretation, context extraction, generation of behavior models, and algorithms designed for collaborative sensing and collaborative processing **[20]**.

When implementing and using these concepts in the field of Ambient Intelligence, certain adaptations should be done considering the context of use. The context of use may favor certain technologies, approaches, or concepts over others according to the main goal of the application or the current task (in this case the Ambient Intelligent system may have to change its configuration for each task, thus the importance of self-configuring capability in this case), for that reason we reckon that there is no absolute best architecture for the perfect Ambient Intelligent System. For example, in **[45]** the authors show that techniques from a patient monitoring application, which are designed mainly to improve the quality of life of patients and/or the elderly by assisting them and detecting any abnormal behavior or accidents, are considered as intrusive in many other contexts where privacy is a priority.

3.1.3.2. **Human-centered compting**

Ambient Intelligence is a new paradigm that uses, among other tools, low level technologies, mentioned in the previous paragraph, and combines them to achieve human centricity, which lies at the core of Ambient Intelligent systems. Ambient Intelligent systems use technology to serve their user in whatever form and flavor that best offers the intended experience of the application of interest by the user. In this new field many concepts are being used, such as privacy management, ease of use, unobtrusive design, customized system, self-healing, self-configuring and self-management **[31]**.

In **[46]**, human-centered computing is defined as a set of methodologies that are applied to any field that uses any form of devices that have computational capabilities. On the latter devices ought to be installed applications allowing humans to directly interact with the whole system. Human-centered field can be divided into three main areas of research: content production by users, analysis of data, and human-computer interaction. Human-centered computing main aim is to facilitate for non-technical users of intelligent environments the translation of conceptual ideas of the functionalities of computerized environments (and/or appliances) into concrete designs. A good example for that is Pygmalion, which is a visual programming system that was implemented on an interactive computer with a two-dimensional graphical display **[47]**. The technique was further developed by **[48]**, where it was called 'programming by example'. The proposed approach allowed non-technical users to program distributed computing platforms for intelligent systems.

In order to achieve the vision of human centricity in Ambient Intelligent Systems a number of novel concepts and methodologies are used, in particular the deconstructed appliance model, which is a new networked service aggregation model developed by **[49]**, and defined as the

process of decomposing, then reconstructing, traditional appliances into more elementary functions that are network accessible. For instance a television set can be decomposed into more 'atomic' network services, such as display, audio transducer, etc. These services can be shared between devices (display can be shared between a TV and a computer). They also can be combined with other services allowing people to re-configure them into novel combinations. Such technique creates what the authors call 'personalized soft-appliances'. The concept of meta-appliances/applications is introduced. Another concepts used are dComp **[50]**, which is an ontology to support decomposition, and Pervasive Interactive Programming **[51]**, which is a tool for programming coordinated behavior in rule-based network artifacts. User evaluation of this methodology demonstrates that users find the system both simple and enjoyable to use.

Beyond the comfort that Ambient Intelligent systems provide their users, these systems play a more 'serious' role when it comes to the well-being of people suffering from physical impairments. The authors in **[20]** reckon that the intelligent human computer interfaces, by facilitating web browsing, sending emails, posting messages, text or image editing, creating animations etc., can significantly improve the whole experience of communicating and participating in the information society for physically impaired users. Technically, assistive software process real time input from advanced camera-based interfaces to help physically impaired users execute in an easier way. The authors also reckon that camera-based interfaces are preferred by physically disabled users because of the fact that they do not require much physical interaction. Another factor that makes such interfaces preferable is the fact that they are customizable, which facilitates taking account of specific particularities of different physical impairments.

The adaptability and customizability nature of Ambient Intelligent systems allows users of these systems to create their own smart environments that know their preferences and understand their speeches and gestures. This causes new challenges when the ambient environment is a shared space between different users with different habits and preferences. Among these challenges we can cite the automatic analysis of data of multi-party interactions (with the system or between users themselves) that come from sensors **[20]**. The interactions can be verbal or nonverbal, such as gestures, postures, activity, visual attention etc. In order to extract information from nonverbal cues Ambient Intelligent systems have recourse to cognition, social psychology, and social behavior analysis techniques. These techniques are adapted to be used in smart spaces equipped with sensing devices such as microphones and cameras. Many applications use these techniques to enhance remote conversation between users and to recognize social behavior of the occupants of the smart environment.

To push the technology even further, wearable system for automatically sensing, inferring, and logging a variety of human physical activity have been introduced in **[52]**. The data collected from such systems can be exploited by Ambient Intelligent systems to analyze users' behaviors outside the smart environment, which enrich the data collected about the behavior and preferences of the user and improve the overall performance of the Ambient Intelligent system.

This concept of mobility and content access (and upload) from anywhere raises new challenges **[20]**. For instance the Ambient Intelligent system must allow the streaming of heavy multimedia content such as music, pictures, video clips, games, etc. via many mobile devices and/or appliances in the environment. Not only that but the system should do so without overloading the Ambient Intelligent system available communication means; especially that the latter are necessary for more important system tasks. Moreover, in some cases, the content of the information must be altered to suit the device, the context, or the user profile; hence the introduction of new content delivery mechanisms that are based on identifying and considering the current context while delivering the information. This method is claimed to emphasize user-driven design. In fact, when shared content environments are being set up and configured in an

Ambient Intelligent system, the users' roles exceed that of content consumers to content producers and co-designers.

3.1.3.3. **Multi-modal human interaction**

In the same scope, which is using advanced human-system interaction in order to make the use of the system as natural as possible; works from the field of multi-modal human computer interaction are being re-used in the Ambient Intelligence field. The goal is to create specialized dialogue systems for interaction with the user **[15][16]**. These specialized dialogue systems, using heterogeneous sensors' data, collects information about the user and the context, then applies the proper dialogue management strategies, user models and contextual information to link the user with its environment **[20]**. The dialogue system must also properly handle the case when the interaction is initiated by proactive Ambient Intelligent environments. A possible scenario allowed by such specialized dialogue systems is using speech or gesture to turn on and off an appliance in a smart home. From other works done in multi-modal dialogue between Ambient Intelligent systems and users, more precisely in the representation of information field, we cite **[53]**, which aims at using several communication modalities to produce the most relevant user outputs, **[54]** a context-aware assistance device for the blind, and **[55]**, which provides a multimodal output specification and simulation platform for the design of an output multimodal system. The proposed framework helps overcoming difficulties that are mainly due to the richness of Ambient Intelligence interaction contexts. As a matter of fact, the contextualization of the interaction becomes mandatory because of the diversity of environments, systems and user profiles. Historically interactions had to be adapted to a given application and for a specific interaction context. In an Ambient Intelligent system, the interactions have to be adapted to a multitude of different situations and a context that is in constant evolution and changes **[56]**. This diversity of the interaction context emphasizes the complexity of a multimodal system design. It requires the adaptation of the design process and more precisely the implementation of a new generation of user interface tools. These tools should help the designer and/or the system to make the proper choices for the interaction technique or modality to be used in a given context at every moment with each user profile or state.

3.1.4. Smart Environments

Smart Environments (SmE) are spaces that are equipped with Ambient Intelligent systems. They are the direct application of Ambient Intelligent technologies so they are environments where we have the concept of "disappearing computers" **[57][58]**.

From a hardware point of view, Smart environments are places that have been enriched with technologies such as sensors, processors, actuators, information terminals, and other devices interconnected through a network, in order to enhance services provided to human occupants of the environment.

Technologically, an Ambient Intelligent environment is a multi-sensory environment, generally supported with embedded computer technology. Such environment can capture and interpret what the user is doing, maybe anticipating what the user is wanting, and therefore the environment can be pro-active and re-active; that means capturing what is going on for later use, or acting as an environment that assists the user in real-time or collaborates with the user in real-time. The boost in use of ubiquitous computing technology will help spread (and hide) computing and communication capabilities all around us. That is, creating smart environments in our daily work places, in our home environment and in our recreation spaces, equipped with intelligent perceptual competence helping us to profit from this technology. The miniaturization of the technology has made it possible to integrate, in a concealed way, a lot of devices in our

private or public environment. We rely on these devices for our everyday activities. A lot of these devices and household appliances have already become so integrated into our daily routine that we use them without consciously thinking about the technologies and the 'intelligence layer' behind them. In fact, now we have processing and computational capabilities in almost all household appliances, from refrigerators, heating systems and even toys and learning devices.

This processing capability makes it possible to add a layer of intelligence (reasoning capacity) around us. The same computational capabilities can be found in modern cars that are equipped with embedded sensors and actuators that assist the driver. Another application of smart environments in public places is tracking devices allowing the detection of particular situations in crowd behavior. Like for example detecting a dangerous situation where the number of person becomes dangerously big in a subway station, or facilitating the evacuation of a crowd in emergencies **[59]**.

With the continuing progress in computing powers in smart environments, our lives are enhanced furthermore, whether it is by anticipating when an action should be done (for instance turn on lights), facilitating transport commuting, or helping to take care of patients in hospital. To end up with "smart environments" from these computing devices they have to be coordinated by intelligent systems and mechanisms that properly integrate and make 'intelligent' use of the available resources in order to improve the lives of their users. Joining together all these technologies and concepts helps the creation of smart environments. A Smart Environment is defined by **[60]** as "a digital environment that proactively, but sensibly, supports people in their daily lives".

The most known applications of Ambient Intelligence are Smart Homes, hospitals, cars, classrooms, offices, malls, etc. Ambient Intelligence enhances the global behavior of such a system by providing high level functionality which provides an added value to the typical services expected in a specific environment. This is done by continuously analyzing, in real-time, the events occurring within the smart environment and taking the proper actions and providing the proper services to the occupant of the smart environment. There is a wide range of services that an Ambient Intelligent environment can provide. Typical examples are services related to guaranteeing occupants safety in smart homes. Smart homes also help assisting people with physical impairments or advanced age to safely perform some tasks. Equally, cars can be transformed into smart environments to assist drivers in difficult conditions, classrooms can be equipped to enhance the teaching-learning experience and offices can be supplemented with technology to support effective workgroup collaboration. By Ambient Intelligence we refer to the mechanisms that control the behavior of the environment.

Next we present some examples of smart environments.

3.1.4.1. **Smart homes**

Smart homes are the most known application of ambient intelligent systems. A smart home is a residential setting equipped with a set of advanced devices (sensors and actuators) designed for improving the safety, comfort and quality of life of the home residents, care delivery, remote monitoring and early detection of problems or emergency cases. To do so, information and communication capabilities are integrated into the smart homes devices so these devices can deliver non intrusive services to the users. In **Figure 26**, we see an example of a smart home where computing modern devices disappear in the background replacing old home devices.

Figure 26. Example of a smart-home where computing devices disappear into the background [61]

These systems are user-centered as they are designed to address the needs of individuals. Many smart homes are designed in a way that focuses on the creating an environment that maximizes the productivity of its inhabitants and minimizes operation cost **[62]**. In that sense Smart Homes can be used to encourage better lifestyles by comparing trends, or action patterns, recorded from the activities of the occupant over a longer period of time, against targets set by the house occupants. For example we can imagine achieving economical targets through a rational use of energy, or entertainment goals by providing personalized television or music services depending on the types of activities and time of the day that they are requested. In order to achieve this goal, the house must be able to predict reason about, and adapt to the context and to its inhabitants' preferences **[63][64]**.

Recently a new field have emerged called Ambient Assisting Living (AAL) **[70]**. The field of AAL is open and changing; hence AAL applications are very technologically rich and extensible. Moreover, AAL applications are trans-disciplinary. For instance, they can mix automatic control techniques with modeling of user behavior. The field also covers home-based healthcare monitoring systems, which are described more in the next paragraph.

3.1.4.2. **Smart hospitals and healthcare monitoring systems**

Like in most fields nowadays, the equipment in the medical field is becoming more and more computerized. In fact modern hospitals have been transformed into sophisticated medical facilities where diagnostic gear, analytical equipment, drug dispensing carts, computerized physiotherapy, patient infotainment terminals, multi-parameter patient monitoring devices, etc. are based on computers.

Continuing on the same path of seeking a better and more effective care with shorter hospital stays, modern hospitals are constantly evolving towards smarter and more connected hospitals networks where everything is managed using computer-based platforms. This facilitates providing and managing care. The latter task is made easier by modern and very efficient computerized systems to access medical data. An example of this is the mobile point-of-care (MPoC) presented in **[65][66]** (see **Figure 27**). The idea is to use tablet computers with all of standard patient care tasks. These computers allow medical staff to access the most recent patient data (blood test results for instance), at all time. This solution improves nurse/doctor communication and by consequence the workflow of the hospital staff. From the patient point of view, this solution improves the overall experience and shortens the hospitalization time.

Figure 27. Example of a Mobile Point of Care

In general modern medical devices have touch-screen interfaces, and the recent embedded platforms, designed specifically for smart healthcare systems, provide the hardware requirements of scalability and performance **[67]**.

Note here that dependability is a critical aspect of the system. Relying on failed services in such a context is hazardous, hence the importance to equip such systems with automated intelligent diagnosis mechanisms **[68][69]**.

It is to be noted that there are smart healthcare systems that are designed to be installed at homes creating smart home health care systems **[71]**. Such systems adapt different techniques and technologies, which are not specifically designed for health care such as sensor networks and distributed vision (camera)-based systems **[72]**, to make them useful in the context of home based health care monitoring system **[73]**.

3.1.4.3. **Smart industrial plants and factories**

*"A smart factory is characterized by several paradigms. All objects - machines, field devices and products are smart. That means that they have sufficient computing power and communication capabilities to allow for autonomous operation."***[74]**

The latter definition came as a result of the continuous research work, mainly, by German industrials. In fact smart factories started as computer-integrated manufacturing systems in the 80's. The idea was to make the industrial manufacturing technologies better by experimenting with new industrial applications. The progress in technologies, especially information and communication technologies led to the miniaturization and made the systems more intelligent. In the beginning such technologies were used to satisfy users' preferences. The idea of smart plants is to use these technologies in the context of industrial environments in order to assist in mass production tasks. The idea is to make these tasks perform more efficiently and 'intelligently'. In **[75]** the necessity of Ambient Intelligence for industrial use is discussed.

We notice that in industrial environments, old methods are still being used even though technology is available, mainly because of costs and risks of changes in industrial environments. New types of factories are created as feasibility demonstration and test bed for the concept of 'smart factory'. Ubiquitous technologies are tested in these factories under realistic conditions. Smart factories are tested for adaptability to meet the demands for variable products and production methods.

The users are central to this new technology. In fact it is designed, not to replace the factory workers but to optimally assist them in their tasks. It also assists administrators to supervise production processes using the central control desk **[76]**. The latter, as in most traditional factories, allows monitoring the production process. However by using advanced simulation

technologies combined with ubiquitous devices. This makes it easier to switch between real world applications and their virtual reality counter parts, which allow the administrators to easily plan new production processes, change them or adapt them according to clients needs. This makes new smart factories easily modifiable, expandable and very 'flexible' **[77]**.

In this settings wireless technologies are used to improve agility and gain time required to connect devices with wires. In particular RFID chips **[78]** are used to identify products, wireless communications are set between devices and smart-phones are widely used **[80]**.

It is important to note that RFID chips are widely used in other application domains of Ambient Intelligence in order to improve the context awareness of ambient systems by locating different components of the environment. For example, in **[79]** RFID tags are used to locate different objects around an interactive table in order to adapt its behavior to different possible situations around it.

In traditional settings actuators and sensors used analog control signals to communicate, however with modern wireless technologies, such as Wifi, and microprocessor technologies that have been integrated in the actuators and sensors, these components can exchange more complex information and can handle more various situations locally.

A demonstration factory, for producing and bottling liquid soap, was built in Kaiserslautern, Germany **[81]**. The smart factory went into operation and served as a research testbed for smart technologies and for the integration of the smart principles.

In the industrial world, one of the companies that are adopting the smart factory philosophy is Hareon Solar Corporation. It uses the paradigm to improve productivity across its solar photovoltaic (PV) cell manufacturing operations in China.

In the years to come, further development is expected from advances in ubiquitous technology, the increasing of Internet based technologies, the desire for safer factories or the creation of new standards and specifications **[82]**.

3.1.4.4. **Smart transportation systems**

The concept of intelligent transportation encloses a huge range of systems and applications. Smart electric vehicle (EV) charging, citywide traffic monitoring, real-time traveler information, transit signal priority, centralized vehicle fleet management, the recent Tesla electric car and the infrastructure imagined around it allowing the availability of charging stations for different routes, etc. can all be classified as forms of intelligent transportation systems. What makes them smart is the use of embedded intelligence to connect vehicles to each other and to the infrastructure, as well as to central operational sites.

Transportation systems are also considered smart when they are applied to achieve smart policy goals in the urban environment, such as enhanced mobility, lower emissions, reduced fuel consumption, improved safety, or economic competitiveness **[83]**.

3.1.4.5. **Smart museums**

This is ambient intelligence applied to the area of museums to create smart museums. The idea is to guide the users in experiencing art in an interactive manner using augmented reality techniques and other media technologies. These technologies are usually designed, by artists or professionals, for a particular artwork or exhibition **[84]**. In this context the methods, tools and technologies developed in ambient intelligence domain become part of the infrastructure of museums. For that, the context, the domain, its inhabitants (visitors, participants, users, etc.), objectives, and activities ought to be understood by the system. According to characteristics such

as the duration of a visit, the sequential or non-sequential behavior, the selectiveness, the number of stops, proximity to the exposition items, etc. different visiting methods are identified in **[85]**. Those who would not follow the designated routes of a typical visit are called the *grasshoppers*. Those who take their time studying the items in the exposition are called *the ants*. Those who are only interested in some items are classified *butterflies*. And finally those ho would glance quickly and superficially are called *the fishes*.

In MIT's Museum Wearable project **[86]**, a different classification is made distinguishing between busy, greedy and selective visitors. The idea is to anticipate, support and influence the behavior of different types of visitors, for instance by drawing their attention to items in the exposition that may interest them.

The technologies can also be used to allow remote visits in real-time. In the HIPS project **[87][88]** visitors are given handheld guides through whish they can bookmark moments of their visit, allowing visitors to experience the visit again, share it with others and help plan a next visit.

Other works have been done to improve the experience in the museum making it 'smarter' and more enjoyable. For instance, in **[89]** the authors used virtual and augmented reality technologies in order to simulate the human museum experience with both autonomous agents and agents representing humans. In **[90]**, the authors emphasized the domain knowledge (of museums) via a better modeling of events and actions in smart environments. In **[91]**, the authors used a semantic Web framework thus improving the contextual information. This allowed the visitor-oriented framework's service to identify relevant resources to the visitor and to enforce user-specified privacy preferences about what information to be shared with other users.

3.1.4.6. **Smart campus**

The goal of a smart campus is to have a view on the activities that are happening at the university campus in order to offer adapted services to the users. A location and context aware smart campus information system is presented in **[92]**, and the ubiquitous computing architecture and the interface design for this smart campus framework are detailed. In such environments available services allow navigation using proximity detection and information posting for news, activities and schedules using Bluetooth and Short Message Service (SMS).

Other works have been done for smart campuses, for instance in **[93]** the MyCampus group developed an open Semantic Web infrastructure for context-aware service provisioning using the concept of e-Wallets, which provides answer to users' demands for context awareness whilst maintaining their privacies. Similar work is done in the iCampus project **[94]**; however its main motivation is to improve the mobility of blind or partially sighted students within a campus. That is why it emphasizes on the mobility of the provided services, their quality and their ease of use by visually impaired students.

To sum up we can say that the definitions and examples of Ambient Intelligence given above share a special characteristic which is the need for a 'sensible' system, this means a system with reasoning capabilities and intelligence. The definition reflects an analogy with how a trained assistant, (e.g. a nurse or a technician), typically behaves. The assistant will help when needed but will not intervene and respect the user personal space when its services are not needed. Being sensible demands understanding the user, learning (or knowing) her/his preferences, and being able to exhibit understanding toward the user's mood and the current situation. In this thesis, we reserve here the term "Smart Environments" (see for example **[95]**) to emphasize the physical infrastructure (sensors, actuators and networks) that supports the system and not the software part where the intelligence and reasoning capabilities reside **[96]**.

3.2. Technologies

One goal of our work is to provide higher-level abstractions of low-level concepts of ambient intelligent systems, in order to facilitate the design and implementation of a self-diagnosing Ambient Intelligent system. For that we do not give a complete list nor a very detailed description of the low-level technologies used in Ambient Intelligent systems. Nevertheless, in this section we present some of the most used hardware components, we detail their hardware specifications, technologies, and particularities.

3.2.1. Controllers

A lighting controller is an electronic device used in building to control the operation of one or multiple light sources at once. Majority of lighting controllers can control dimmers which, in turn, control the intensity of the lights. Other types of controllers can also control lighting, according to specific scenarios. Lighting controllers communicate with the dimmers and other devices in the lighting system via an electronic control protocol (DALI, DMX, ZigBee, KNX, etc.). The most common protocol used for lighting today is Digital Addressable Lighting Interface which is commonly known as DALI. Controllers vary in size and complexity depending on the types of buildings (from small residential buildings to big tertiary one). For most of the time the purpose of lighting controllers is the same: to combine the control of the lights into an organized, easy-to-use system, and to reduce lighting energy consumption.

Controllers are used to control the functioning of actuators. In addition to simple controllers (ex. Automatic On/Off light switch equipped with a presence sensor), there are more advanced controllers that can offer advanced control of actuators (ex. Light dimmers **[97]** controlling the intensity of lights), and can even offer the possibility to be programmed to follow more complex specific scenarios. Controllers communicate with the devices via an electronic control protocol. Among these protocols we cite ZigBee high level communication protocols suite **[98]**. Zigbee-based controllers (**Figure 28** and **Figure 29)** can easily communicate with their devices via a Zigbee ad-hoc network. Some applications in an Ambient Intelligent environment include wireless light switches and automatic light dimmers that permit the optimization of light use and reduce energy consumed by the light system.

Figure 28. ZigBee lighting controller: Touch Panel Dimmer Switch (one way)

Figure 29. The U600LF model of ZigBee relay control; to be placed between power source and lighting device to control it wirelessly.

Other than ZigBee devices, there are other lighting system controllers such as DALI (Digital Addressable Lighting Interface), DMX, KNX, etc.

3.2.2. Actuators

Actuators are the means with which ambient systems act upon their surroundings. They convert electric orders into physical actions. Actions of actuators can be emitting light, sound or heat; physically affecting other objects by moving, heating, cooling, destroying, displaying information, and any other possible action that changes the state of the environment. In more general terms actuators are system components whose states can be changed by orders from the ambient system via controllers; by changing their states they act upon they surroundings.

In our models the entity "Actuator" represents in fact actuators (as defined in this paragraph) and their corresponding controller(s) (as defined in the previous paragraph) as well. This means that we suppose that actuators and their controllers are a single entity and that there are no errors possible in the connection between the actuator and its controller.

3.2.3. Sensors

Sensors are the components responsible for keeping the system aware of the state, and changes of states, of its surroundings. Sensors are responsible for measuring the real world conditions. A sensor converts a measurable physical quantity into a signal that can be processed by the system.

For example in the case of light, a light sensor detects light intensity levels (*Definition: Light intensity measured at a particular position, also called Illuminance and measured in lux – See* **Annex-A** *for detailed definition and references*), the existence of light (or its absence), a variation of light state (for instance a transitions from the state On to Off) a variation from light values (for instance a sudden dimming of light), etc. Then it converts these different entities into data that is manageable by the system.

In the case of heat, sensors report current temperature, detect an increase (or decrease) in temperature, detect fires, etc.

Sound sensors might simply detect the existence of noises; others might go as far as to sample the sound signal.

There is a variety of other sensor types such as presence sensors, pressure sensors, tactile sensors, position sensors, wind speed sensors, etc.

In **[99]** the authors define and describe some of the particular characteristics that affect their performance. Below are some of these characteristics (The Figures are also from **[99]**):

- Sensitivity: the minimum input of the measured physical entity that can create a detectable change in output. A very sensitive sensor is a sensor that measures very small changes in the measured physical entity.
- Range: the maximum and minimum values of the physical parameter that can be measured.
- Precision: the degree of reproducibility of a measurement. An ideal sensor would output exactly the same value every time it is given the same input. In reality sensors output a range of values distributed in some manner relative to the actual correct value.
- Resolution: the smallest change in the input that can be detected by the sensor.

- Accuracy: is the maximum difference between the actual value and the value that can be deduced from the output of the sensor. It can be expressed either as a percentage of full scale or in absolute terms. It is from this characteristic that we will deduce the tolerance value that will be used in the models of our proposed fault detection and diagnosis framework.

- Offset error: the output value when, theoretically, it should be zero. In other cases it is the difference between the actual output value and a calculated output value under particular conditions.

- Linearity: is a mathematical formula that measures the extent to which the actual curve of measurements of a sensor is different from the ideal curve. **Figure 30** shows an example of such relation between the ideal curve and the curve actually measured.

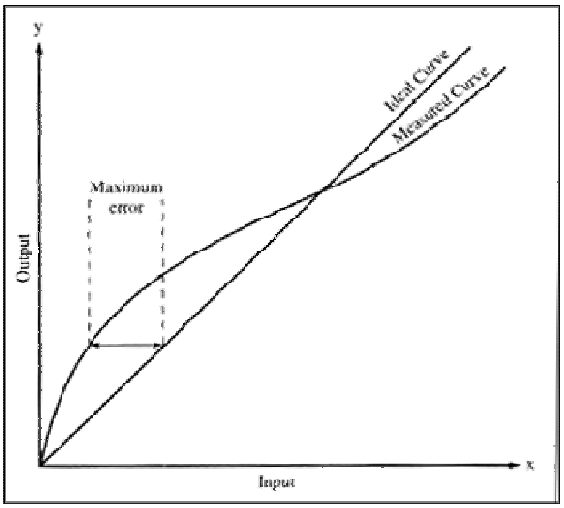

Figure 30. Ideal versus measured curve showing linearity error

- Hysteresis: the measure of the ability of a sensor to follow the changes of the input parameter regardless of different previous states and/or values. **Figure 31** shows a typical hysteresis curve. When approaching a fixed input value (for example the point noted B in the curve of **Figure 31**) from a higher value (like point P), we obtain a certain output value. This value is different from the value we obtain when approaching the same input value from a lesser input value (like point Q or zero). Note that the input value B can be represented by $F(x)_1$, $F(x)_2$, or $F(x)_3$ depending on the immediate previous value. This is an error due to hysteresis.

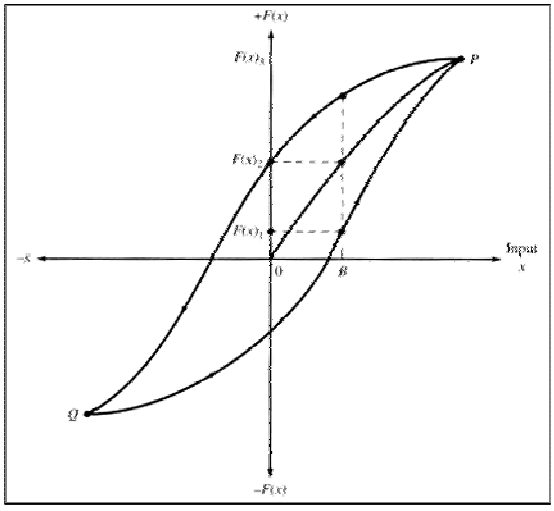

Figure 31. Hysteresis curve

- Response Time: is the time necessary for a sensor to change its output state (to the correct value) in response to a change in an input parameter. The period of time necessary to make the change is called the response time (noted *T*). In other words the response time is the time required for a sensor output to change from its previous state to settle on a final value within a tolerance band of the correct new value. This term can be seen as similar to that for a capacitor charging through a resistance. In **Figure 32**, the curve represents the response time following an abrupt positive change of the input parameter of the sensor (Note the tolerance band). The form shown in **Figure 33** is a decay time (noted T_d) in response to a negative change (or absence of) of the input parameter. Decay time is to be distinguished from response time as they are different for many sensors.

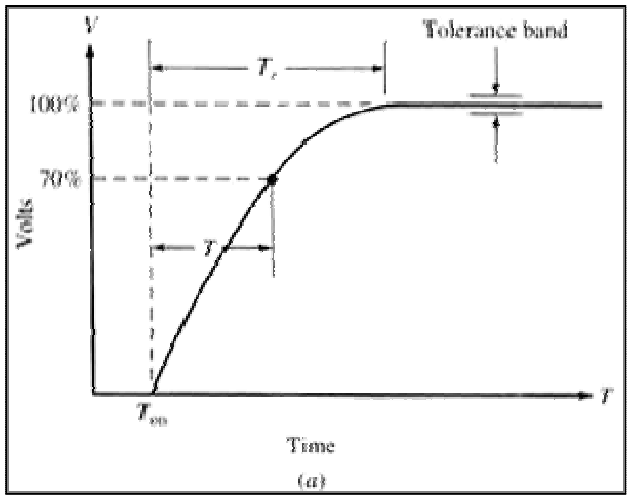

Figure 32. Sensor rise-time definition

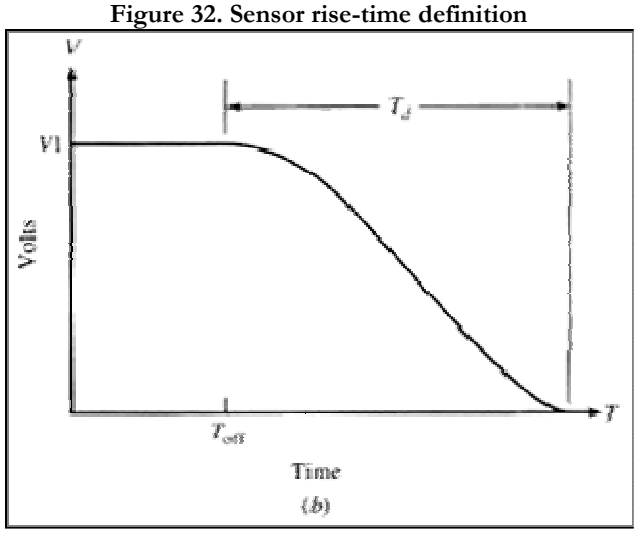

Figure 33. fall-time definition

- Dynamic Linearity: The dynamic linearity of a sensor is a measure of its capability to follow abrupt changes in the input parameter. Many characteristics are important to determine the value of the dynamic linearity. These characteristics are Amplitude distortion, phase distortion, and response time. To calculate the value of the amplitude response in a system of low hysteresis, we use:

$$F(x)=ax+bx^2+cx^3+dx^4+\ldots+K$$

In this equation, the term *F(x)* is the output signal, while the *x* terms represent the input parameter and its harmonics (in a wave form signal, a harmonic is a component frequency that is an integer multiple of the signal main frequency; for instance if the frequency is f, the harmonics have frequencies 2f, 3f, etc.), and K is an offset constant.

The harmonics are important when the error signal harmonics of the sensor fall into the same frequency bands as the correct harmonics produced by the sensor. Continuous waveforms are represented by a Fourier series of a fundamental sinewave and its harmonics. In the case of nonsinusoidal waveform (like time-varying changes of a physical parameter), harmonics can be affected by the action of the sensor. The harmonics can be deduced based on the nature of the nonlinearity of the calibration curve (**Figure 34**). On the example in **Figure 34a**, the calibration (dotted) curve is asymmetrical. We can conclude that only odd harmonic terms exist. We can assume that the ideal curve can be expressed with:

$$F(x)=mx+K$$

The equation for the symmetrical case becomes:

$$F(x)=ax+bx^2+cx^4+\ldots+K.$$

Figure 34b introduces another type of calibration curve where the indicated values are symmetrical around the ideal *mx+K* curve. In this case,

$$F(x)=-F(-x)$$

and the form of the equation becomes:

$$F(x)=ax+bx^3+cx^5+\ldots+K.$$

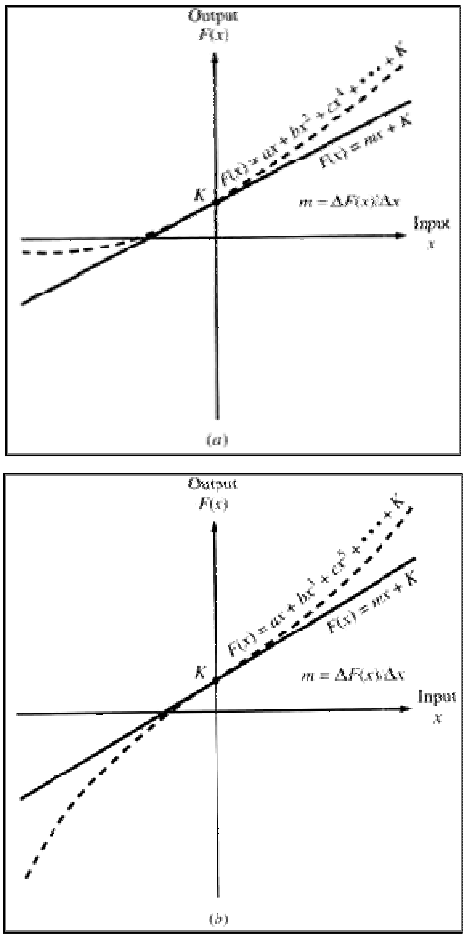

Figure 34. Output versus input signal curves showing (a) quadratic error; (b) cubic error

There are other characteristics that, even though they might be the basis for the choice of a certain type of sensor over another one for the experiments, are not thoroughly discussed in this work, such as cost, compatibility with other devices, used communication protocols, etc.

3.2.4. Sensor Networks

A sensor network is a technology consisting in installing distributed autonomous sensors in an environment in order to monitor some conditions that are detected by the sensors in that environment, such as temperature, humidity, etc. The particularity of the technology consists in the cooperative manner different sensors of the sensor network exchange their data and transmit them to a central location.

Ambient Intelligent systems need to perceive their physical environments before acting upon it. To do so Ambient Intelligent systems rely on a variety of sensors. With advances made recently in the field of "Sensor Networks" in general, and "Wireless Sensor Networks" **[100]** in particular, the applicability of control technology in day-to-day life is expanded, that is largely due the cost-effective deployment of large scale sensor-actuator systems **[101]**. Sensor Networks have become involved into many fields and applications, among which is Ambient Intelligence field and context aware applications.

The technologies used and the techniques applied in research of sensor networks have made it a field of research by itself. The goal of research in this new, and expanding, field is to address particular challenges such as to have efficient resource management, to optimize energy management, better data fusion in order to better analyze the data coming from the different sensors in the sensor network, etc.

These are challenges that are not addressed by our work. And even when fault diagnosis is addressed its goal is to look for faulty sensors within the sensor network based on the analysis of different sensors readings **[102][103]**. Even though, our work detects the existence of faults using sensors' data, we try to isolate the component(s) (not necessary a sensor), or the external factor(s) causing the fault using totally different approach than that used in sensor networks field.

In **[104]**, a wireless sensor network was used in the context of a smart home environment. The wireless sensor network is based on ZigBee communication protocol. The idea is to keep track of the state changes of everyday objects based on the user's interaction with them. However such information is not used for detecting possible faulty states of the objects, it is in fact used to better identify user's activities, current situations, etc.

3.3. Fault Detection and Diagnosis (FDD)

3.3.1. FDD In the field of automatic control: Terminologies and definitions

From the field of automatic control, fault detection and diagnosis systems consist of three main tasks **[105]**

- Fault detection: after comparing the model of the system with the actual system, when a difference is detected between theory and reality, a fault is detected, and the conclusion that something is not working as expected in the diagnosed system is made.
- Fault isolation: by analyzing the symptoms, this task tries to find out the exact cause(s) of the detected fault.
- Fault identification: determining the information that can be concluded about the fault like the magnitude of the fault, its type, location, etc.

Fault isolation and fault identification are usually referred to together as fault diagnosis. Fault detection and isolation are the most important tasks in fault detection and diagnosis systems. Fault identification, even though useful, is sometimes ignored as the effort it requires is not

worth the resulted information. It is to be noted that for the rest of this dissertation we will suppose that diagnosis consists only in the fault isolation task. In many cases, after a fault has been diagnosed an action is decided as for how to deal with the diagnosed fault; the action can vary from displaying a simple alert message to sending an order to the system to correct the specific diagnosed fault. The main goal of that is to protect the system and, if possible, to correct the fault. The signal flow for a typical fault detection, fault diagnosis and fault management system is shown in **Figure 35 [107]**; the diagnosed system is called process.

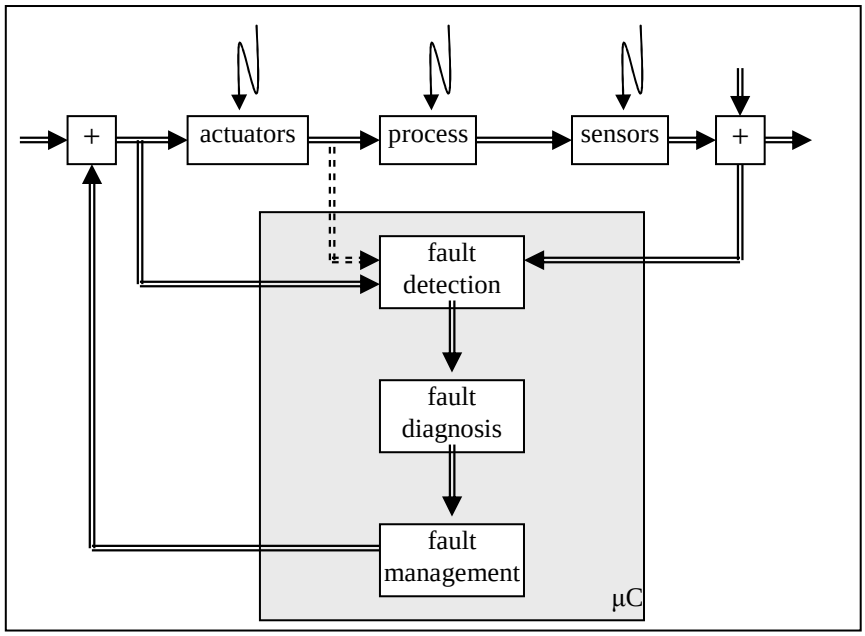

Figure 35. Fault detection, fault diagnosis and fault management system

3.3.1.1. **Fault**

There are different definitions of faults according to the field in which faults are treated. In the field of automatic control, a fault is a deviation of a (at least one) characteristic property of a system from the acceptable, predefined, usual, standard condition. The deviation causes failure(s) and system malfunction.

In the general case a fault can be defined as a deviation of the actual system from the system model, causing it to fail to fulfill its goal(s). To perform real-time fault detection, the system model and the actual system are to be compared at all time in order to monitor any deviation.

3.3.1.2. **Fault types**

Fault types are determined in the fault identification task. Faults are characterized by their form (systematic or random), time behavior (permanent, transient, noise or drift) and the extent (local or global, including the size) of the fault. In reality, fault types highly depend on the system component causing the fault. For instance when there are specification or design flaws in the hardware and/or software part of the system (bugs caused by bad specification or design, coding errors, calculation overflows), the faults caused by that are systematic and not at all random. On the other hand, once the system is running, hardware malfunctions (faults in hardware) are mostly random. Finding out the type of the fault is not one of our centers of interest in our work, in which we suppose that there are no design flaws in the diagnosed system and that all faults are random.

3.3.1.3. **FDD: The offline Vs the real-time method**

Fault detection and diagnosis can be done either *offline*, by simulating the behavior of the system with a model of the system, testing it under different conditions and scenarios, and trying to isolate the critical values or states; or in *real time*, in which case the system model is run in parallel of the real system and theoretical values and states from the model are compared to real values and states from the actual system.

The real time method raises some performance challenges, as computers that are used to simulate the system behavior need to update the modeled system state in a time that is very close to the updates of the real system, which is not an easy task considering the particularities (especially the complexity) of the diagnosed system. For instance, in some cases computers need to have very fast calculation capacities allowing the proper functioning of reasoning tools that exploit the system model to perform diagnosis, in other cases big memory spaces are needed to store big quantities of data that can be used in statistical techniques to deduce faults, etc. This has become less of an issue with modern computers and devices **[108]**. It is however a very important factor to consider when designing the diagnosis system.

3.3.1.4. **Supervision**

In the field of automatic control, supervision is a process consisting in continuously monitoring a system process by inspecting measurable variables and comparing them to a threshold with regard to tolerance values, in order to react to the detection of any deviation (fault) with a counter action to protect the system. This is called limit value supervision. This method is simple and reliable for steady state situations.

However, due to the dynamic nature of Ambient Intelligent systems, thresholds of heterogeneous types (a certain state in a state transition model of a device can be considered as a threshold), and their values are constantly changing and can not be fixed. In fact in an Ambient Intelligence context processes (actions) are changing according to context or use (or reuse) of services by the user. For instance, in a scenario where a radio is controlled by an ambient system to be used as an alarm clock, the radio clock should start at the right time and at the same time play the right station or track. The supervision of devices in this scenario depends more on the context in which the device is used than on the device characteristics themselves. So a supervision method shall be defined for each new context. In addition, even though fault detection is possible with this method, correcting the fault in such complex and changing systems requires more than just the detection of a threshold violation of the values of some variables.

3.3.1.5. **Model-based fault detection method and Fault modeling**

A model-based method of fault detection deduces the existence of faults from changes in the measured values of variables and relations between these variables. To do that the diagnosed system is modeled. In the field of control engineering or automatic control, the diagnosed system (also called process) is modeled with a mathematical model. The latter is called a mathematical process model. Generally, the relations between variables that allow the detection of faults are analytical relations, for instance a fault is detected when the value of a certain feature exceeds a certain limit. In other cases the system model can be in the form of if-then causality rules, for instance the presence or absence of a feature determines the existence of a fault **[105]**.

The proper modeling of the process makes it possible to apply a model-based fault detection method to detect faults. In **Figure 36** we see a general scheme for process model-based fault detection. The mathematical process model describes the relation between the measured input signal **U** and the output signal **Y**. Features, such as parameters **θ**, state variables **x** or residual **r**,

are then extracted. Comparison between these features and their nominal values generates what is called analytical symptoms **s**. Analytical symptoms are the results of the limit value checking of measurable signals, signal or process model fault detection methods and change detection **[106]**. After that these symptoms will be the basis for fault diagnosis.

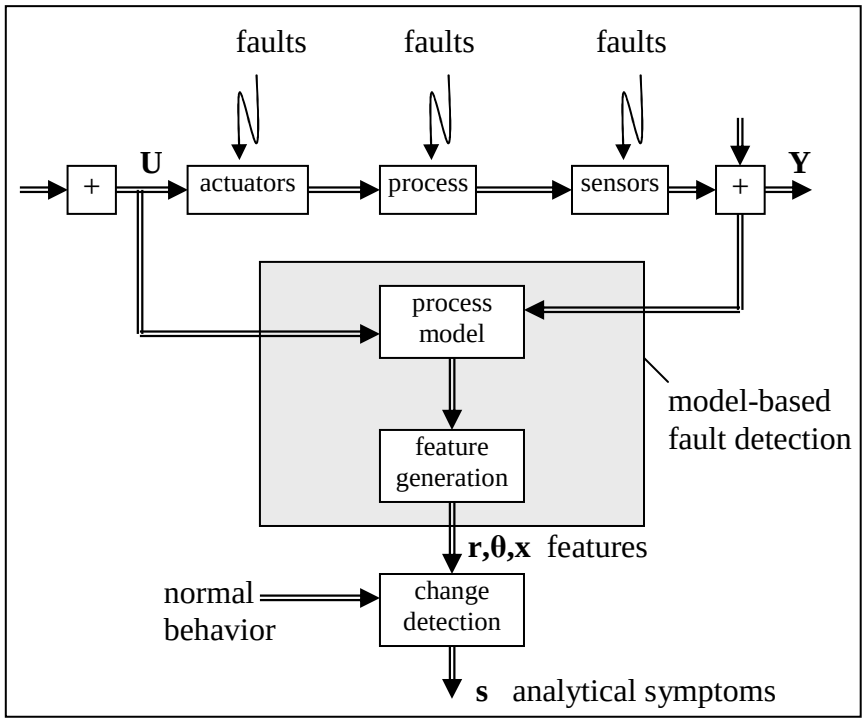

Figure 36. General schema of process model-based fault detection

3.3.1.6. **Fault Diagnosis**

The goal of fault diagnosis is to find out as many details as possible on the detected fault, in particular the cause of the fault. The basis for fault diagnosis is the analytical symptoms **s**. These symptoms are generated via the fault detection task from unusual changes in the features from their normal or nominal values (as showed in the previous paragraph). As a matter of fact, in an actual physical system, faults cause certain events which in turn cause symptoms that are measurable and/or observable. Understanding this cause-effect relationship from faults to symptoms in a system is the basis for creating its diagnosis model. In fact, the diagnosis model is used to deduce faults from symptoms in the reverse way as shown in

Figure 37. This idea is the basis for many fault diagnosis methods. These methods vary mainly according to the available information about fault-symptom causalities of a given system. Two cases are possible:

The first case is when little information is available; in which case *classification methods* are used. The second case is when relationships between symptoms and faults are, at least partially, known; in this case *inference methods* are used.

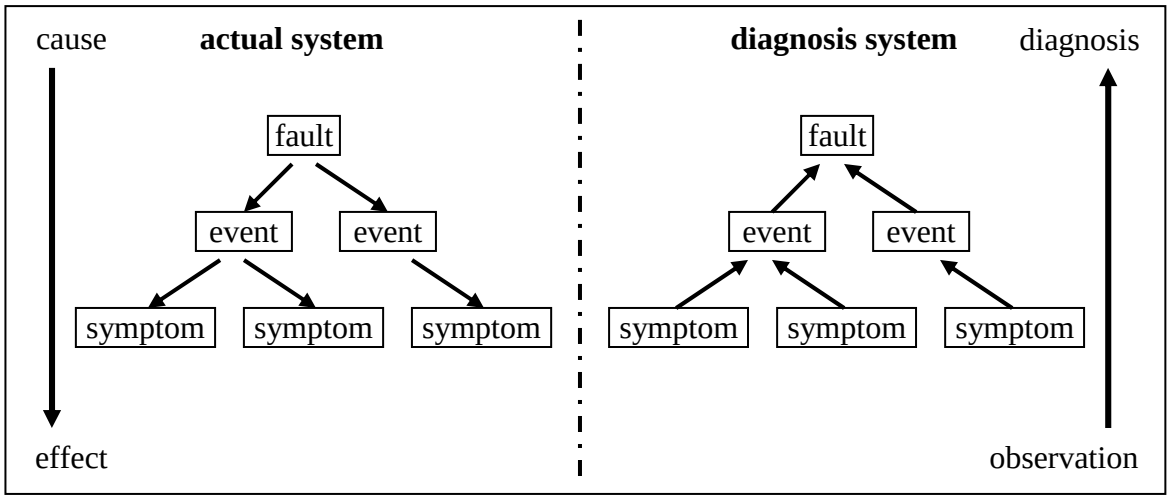

Figure 37. Fault-symptom relationship in an actual system and in a diagnosis system

Classification methods: These methods can be trained with real data from the system without the need for structural knowledge about relations between symptoms and faults. Generally the technique consists in using pattern classification method **[109]** as shown in **Figure 38**. Expected symptoms are represented in reference symptom vectors $\boldsymbol{s_{ref}}$ and are associated with the faults $\boldsymbol{F_j}$ experimentally by learning or training. Observed symptoms (s) are then compared to the reference symptoms to determine the faults by classification. The classification methods are used here as a functional mapping of the diagnosis deduced from the symptom space s to the space of fault measures $\boldsymbol{f_j}$.

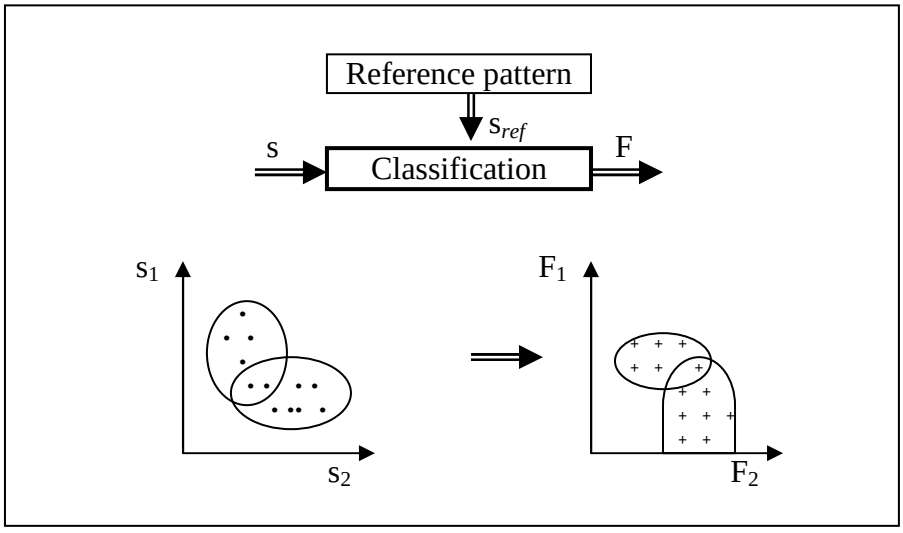

Figure 38. Fault diagnosis with classification methods

The most commonly used classification methods are summarized in **Figure 39**.

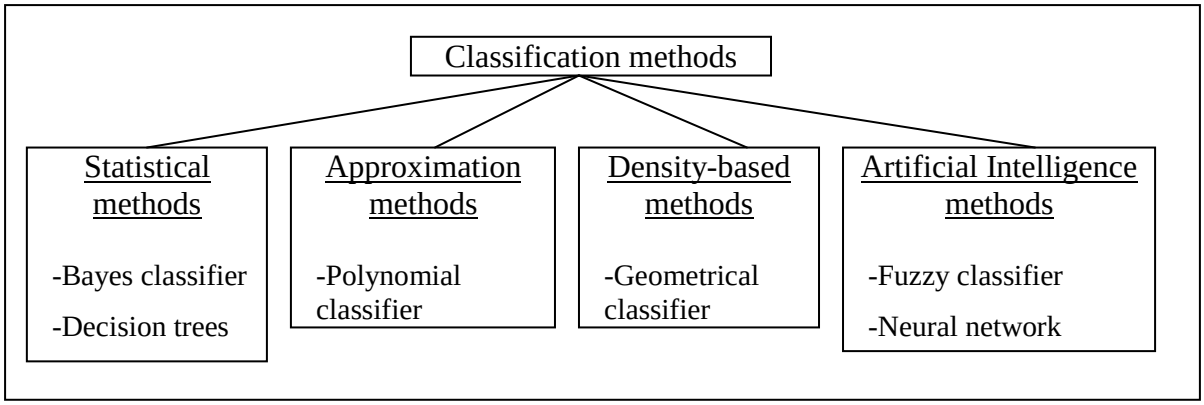

Figure 39. Pattern classification methods

Historically, the statistical methods came first, followed by the density-based methods and general approximation approaches. The artificial intelligence methods were the latest to be developed for diagnosis problems.

Among these classification methods we detail some:

- **Bayes classifier**: belongs to statistical methods of classification. It is the most well-known classification method **[110]**. The method is based on making assumptions about the statistical distribution of the symptoms. A common statistical distribution function is the Gaussian probability density functions **[111]**

$$p(s) = \frac{1}{(2\pi)^{n_s/2}|\sum|^{1/2}} \exp\left(-\frac{1}{2}(s-s_0)^T \sum^{-1}(s-s_0)\right)$$

The procedure consists in determining the maximum likelihood estimations for the constants, the covariance matrix $\sum$ and the centers s_0 (can be found using the mean values of the reference data, recursive parameter estimation methods, or Bayes inference) of the Gaussian probability density function. Building a classification system from the probability density estimations requires class specific densities. It can be shown that a minimum of wrong decisions is achieved if the maximum of a posterior probability $p(F_j|s)$ is selected. The prior probability is the probability that express the uncertainty about p before the actual 'data' is taken into account. This probability is calculated with the help of the Bayes Law:

$$f_j(s) = p(F_j|s) = \frac{p(s|F_j)P(F_j)}{p(s)}$$

The class-specific densities can be estimated from labeled reference data using those data points belonging to 'fault'. Since we are only interested in the maximum of the fault data values, the denominator is not significant (because it does not depend on the function. It only plays the role of normalizing factor and is irrelevant for the comparison).

However, the prior probabilities are important. Provided enough reference data is available, they can be estimated from their frequency of the occurrence in the data set. It is to be noted that in many applications these prior probabilities cannot be determined. In many cases the reference data is created from experiments where the occurrence of the faults can be influenced, in which case, unless experience output a better choice, one should assume that

$$P(F_j) = \frac{1}{n_f}$$

Where $n_f(>0)$ is the total number of faults. This would suggest that all faults have the same probability. The prior probabilities are important for the overall quality of the diagnosis system. In fact if they are carefully selected, they can improve the performance of a diagnosis system significantly. In many cases measurement errors are modeled by a normal distribution. This does not imply that we assume that the measurement errors are normally distributed; it is rather used to produces the most conservative predictions possible given only knowledge about the mean and variance of the errors **[112]**. However, the common assumption of normal distribution can be problematic when applied for real problems. This holds for instance in the case of distributions with overlapping fault areas **[113]**.

If an estimation for $p(s|F_j)$ is represented graphically as a histogram, a more realizable case is then given. Nonetheless, a necessary condition for that is to have sufficient amounts of data. For situations with a lack of enough data points, the histogram methods can be used with variable-size grids. This is very similar to geometric classifiers discussed next.

- **The polynomial classifier [114][115]**: Belongs to the approximation methods. It is when the polynomial classifier is used, instead of the Gaussian functions assumed for the Bayes classification, a special functional approximation is used for the posterior probabilities. For that polynomials are used:

$$p(s|F_j) = f_j = a_{j,0} + a_{j,1}s_1 + a_{j,2}s_2 + \ldots + a_{j,n+1}s_1s_2 + \ldots$$

where $a_j = [a_{j,0}\ a_{j,1}\ \ldots\ a_{j,n}]^T$ are parameters of the polynomials. The classifier is used first to evaluate all polynomials. Every polynomial is associated with a fault class. Then the maximum is found based on the calculated f_j. Each data point corresponds to a polynomial. In **Figure 40** we can see a decision boundary for a two class problem. The decision boundary is calculated by the line of equal polynomial values:

$$p(s|F_1)=f_1(s) \text{ and } p(s|F_2)=f_2(s)$$

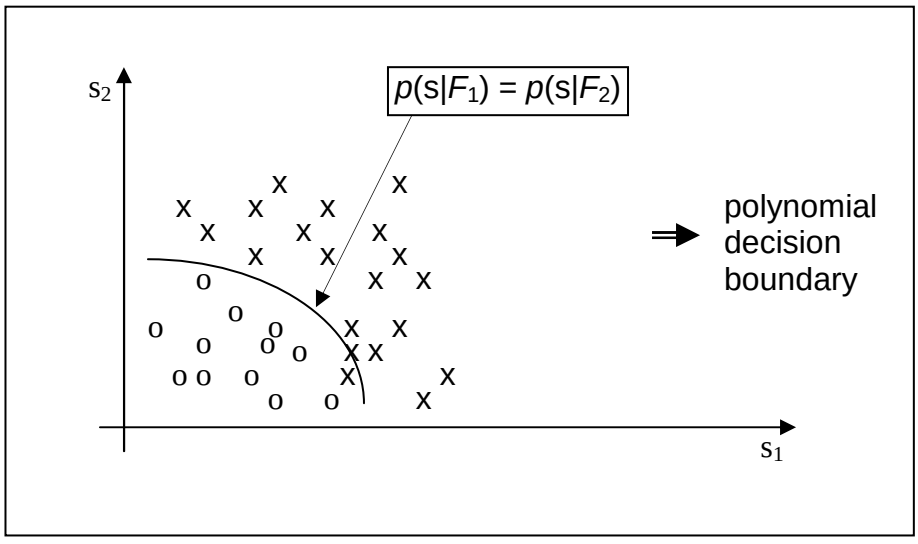

Figure 40. Example of the decision boundary of a polynomial classifier

The difficulty that must be overcome when the polynomial classifier is used is to correctly choose the polynomial terms.

In most applications a complete polynomial is usually (unnecessarily) very large. The challenge here is not to choose a large number of terms because that can easily create an overfitting with bad generalization behavior, and not to chose a number of terms too small because that can make the system not flexible enough to properly distinguish between the classes. To do so, a removal of the nearly linear dependent terms should be done. The removal can follow Gauss-Jordan algorithm **[116]**, or it can follow a transformation like the Orthogonal Least Squares **[117]**, or singular value decomposition **[118]**.

- A special type of **Decision trees [119][120]**: Usually decision trees are not used for classification when we have "little information" about data. However, this special type of decision trees, which is originally from social sciences, is used here to classify data (see binary reasoning paragraph for classic decision tree definition). The system basically relies on answering a series of questions (is the value s_i, of "symptom$_i$", larger or lower than a certain value?). Depending on the answer the next question narrows the answers more and more until the exact answer is determined. A complete decision tree is formed of the collection of all questions. One can picture a whole set of symptoms S_0 of data tuples being subdivided into more sets S_{11} and S_{12} by decision D_1. The two sets are then again divided into more sets forming a tree. Ideally, the splitting is finished if the sets contain solely a single class of data. The class information of the remaining set is called a leaf of the tree. The data is classified in the tree from top to bottom. Starting from the tree root, a new data point is confronted with the tree nodes decisions until it reaches a leaf and is classified according to the leaf's class. **Figure 41** shows such a tree for the distinction of the two faults (or classes of faults) F_1 and F_2 using two continuously distributed symptoms s_1 and s_2. The decisions here are binary but based on a continuous variable. The example also shows that the tree does not have an identical size for all of its branches.

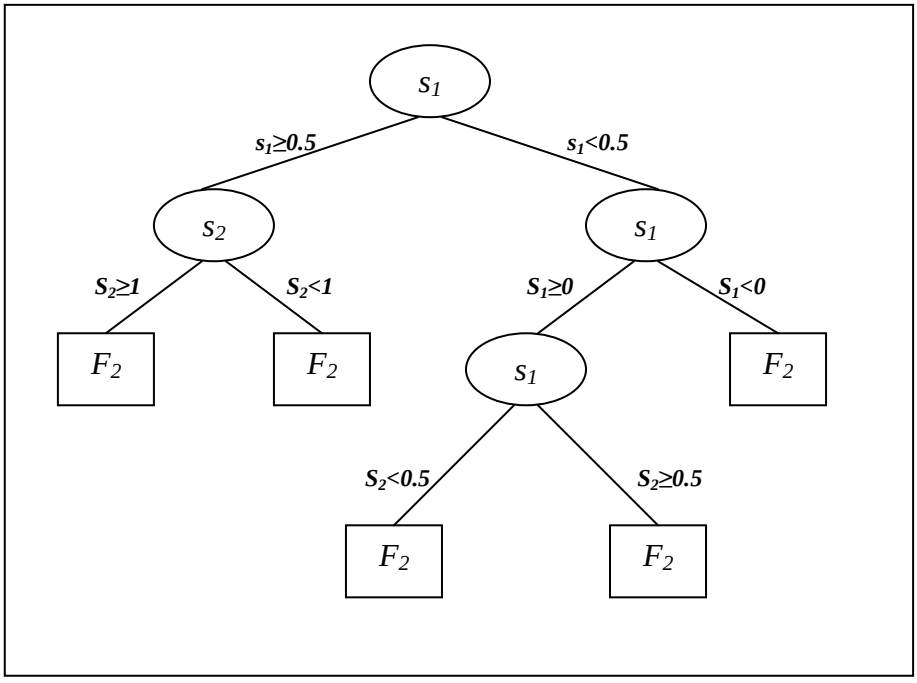

Figure 41. Example of a decision tree for the distinction of two classes F_1 and F_2

The resulting segmentation of the symptom space can be seen in **Figure 42.**

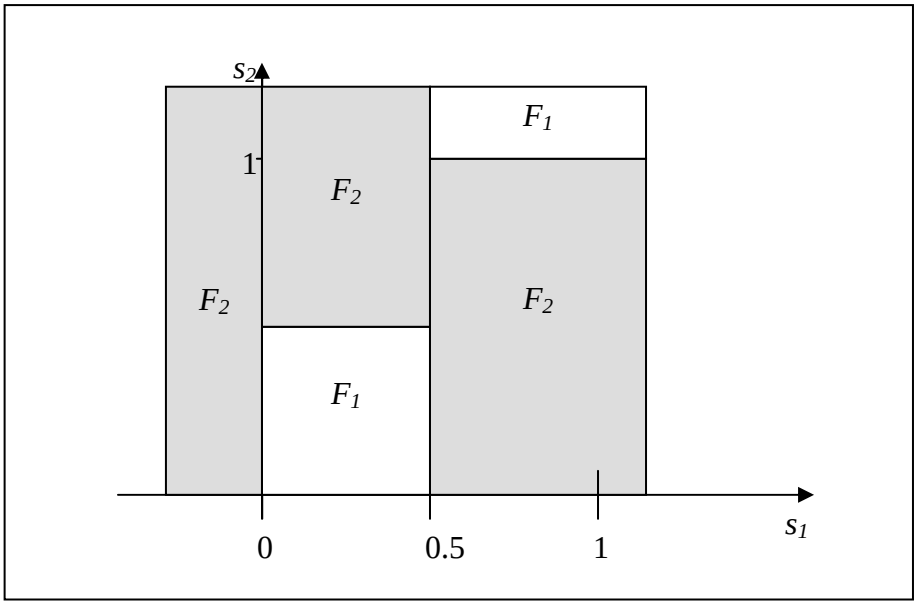

Figure 42. Example of a decision tree for the distinction of two classes F_1 and F_2. Resulting partitioning in a s_1 and s_2 plane, where the symptoms s_1 and s_2 are continuous variables.

- **The geometrical classifier**: belongs to the Density-based methods. The geometrical classifier is based on the calculation of the distance between a data point and a reference data point. This method determines the class membership of the data point. The reference data points are determined by their symptom values $s_{ref,i}$ and their class assignment (for example: class *F1* or class *F2*). Using the Euclidean distance technique, the geometrical classifier is the simplest and most well-known approach. For instance if we want to determine the class of the data point s_j we need to evaluate the minimum

$$\min_i(d_i) = \min_i(\sqrt{\left\|s_j - s_{ref,i}\right\|^2}) \qquad 1 \leq i \leq n_{ref}$$

where n_{ref} is the number of reference data points. As shown in **Figure 43**, the class of $\mathbf{s}_{ref,min}$ that is the closest to $\mathbf{s}_j$ is then considered as the class of $\mathbf{s}_j$.

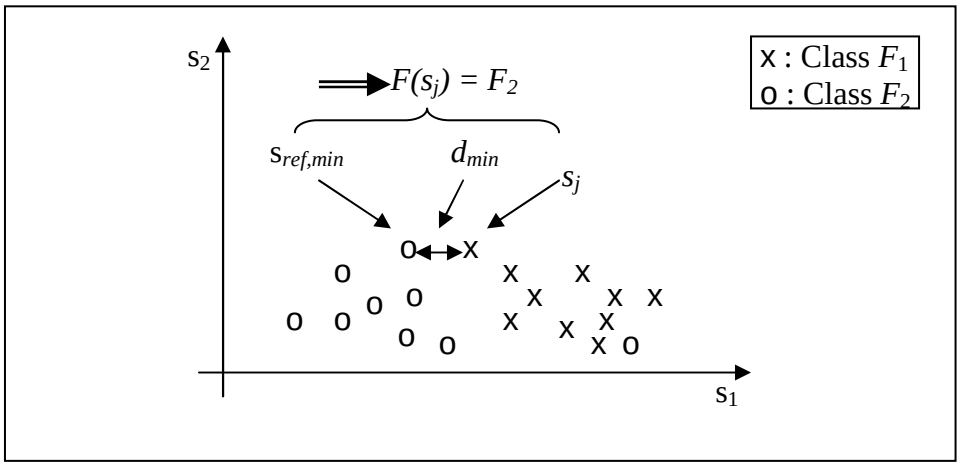

Figure 43. Example of nearest neighbor classification.

This method has some drawbacks. For instance suppose we have only a few reference points and we have overlapping class regions. In that case the resulting decision boundary is not smooth; hence, there is probably a more optimal boundary. This can be fixed when the distance to more than one reference point is evaluated. In that case a voting of the *k* closest

data points is used. This approach is called the *k* nearest neighbor approach. This method can be compared to a local parameter-free probability density estimation. In that case, the class that contains most $\mathbf{s}_{ref}$ is identical to the class with the highest frequency of occurrence centered around **s** in the considered hypersphere. The size of the latter is governed by the reference data density around **s**. Another disadvantage of nearest neighbor classification is the need to store all reference data points. However, modern computation and storing capabilities solves this problem. Furthermore, an intelligent selection technique reduces the number of necessary points to be stored. An example of this technique is the condensed nearest neighbor approach **[121]**.

- **Neural networks classifier [122]**: belongs to the Artificial Intelligence methods. While the Bayes Classifier assumes a special function (Gaussian), the neural networks are designed to match an arbitrary function by reducing an approximate error measure *V*. The mapping function of the network depends on a number of weights *w* that contains the information of the network. Network training is done by adapting the weights to minimize *V*. Neural networks have been shown to be universal approximators, meaning they can fit any function to an arbitrary accuracy, provided their structure is sufficiently large. As diagnosis tools they are trained with exactly the same target values as for instance the polynomial classifier which means that they also approximate the class-conditional posterior probability.

  In the field of fault diagnosis, neural networks are frequently employed: In **[123]** a survey shows that about half of the publications utilizing a classification procedure for fault diagnosis relied on neural networks. Moreover, neural networks provide a means to achieve decent classification results with relatively moderate design effort.

- **Fuzzy classifier [124]**: Also belongs to Artificial Intelligence classification methods. It is very similar to neural networks method in the sense that they both try to determine the transfer function between their feature space and a given class. However, at the contrary of neural networks that can only be initialized in a random state, a fuzzy classifier can be initialized in a particular state (ideally a state close to the correct solution by a designer). This allows fuzzy classifier method to do a faster classification than neural networks.

Inference methods: used when relationships between symptoms and faults are known can be expressed in the form of an if-then rule set. Among inference methods we cite:

- **Predicate logic**: fault trees are an established tool for the visualization of the relationships in reliability and diagnosis **[125]**, representing binary relationships. They are directed graphs showing the fault situation at the top and symptoms and conditions below. Elements of the tree are logic connections, events and symptoms. An example is shown in **Figure 44**:

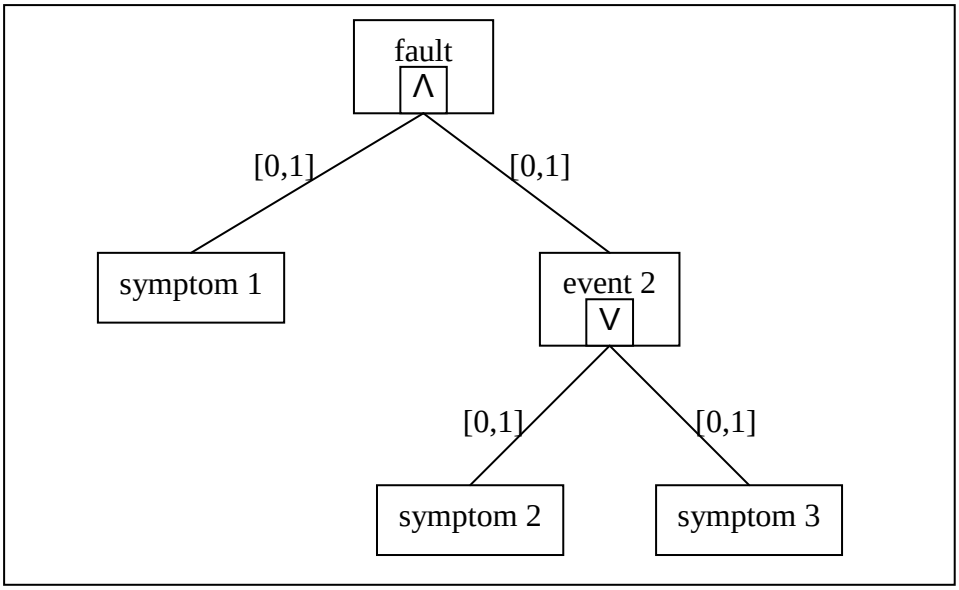

Figure 44. Scheme of a fault tree for binary symptoms

Fault trees are a good and intuitive graphical tool for displaying the binary relationships that will lead to failures. The hierarchical structure supports the human comprehension. They are common for analysis and diagnosis in safety-critical applications. Quantitative failure probabilities (between 0 and 1 as shown in **Figure 44**) can also be derived from the fault trees. They require information about the failure probabilities of the individual elements. By combining these with the relationships from the fault trees it is possible to calculate the probability of a system failure. The fault tree is usually important during the early system design phase in identifying the critical subsystems that contribute most to the system failure probability. Examples of fault trees can for instance be found in **[126][127]**.

The fault trees shown here, for industrial robot, are not used for reliability analysis but rather for the diagnosis. For instance to visualize the diagnostic reasoning from symptoms to faults, one would start at the root "the fault", then by answering a series of logical questions (and/or) about the existence of a symptom or the occurrence of an event we deduce the diagnostic report. It is then necessary to design one fault tree for each of the faults. The leaves of the tree are composed of the available symptoms. The decision: which faults have occurred is ideally a simple binary evaluation of the different fault trees. In most applications, however, this binary representation is not sufficient. It is common that some symptoms are not clearly recognized or that they are uncertain. In this case a straightforward evaluation of the binary decision would not work. A possible strategy is then to use the tree whose complete evaluation could most easily be achieved by changing the status of the symptoms at the leaves.

- **Approximate reasoning**: In the case of rule-based fault diagnosis of continuous processes with continuous variable symptoms, methods of approximate reasoning are more appropriate than binary decisions. Based on the available heuristic knowledge in the form of heuristic process models and weighting of effects, different diagnostic strategies can be applied, such as a forward or a backward reasoning. Finally the diagnostic goal is achieved by fault decision which specifies the type, size and location of the fault as well as its time of detection, **[128][129]**. A review of developments in reasoning oriented approaches for diagnosis was given in **[130]**. As major areas of interest, medicine, **[131][132]**, and engineering **[133]**, can be observed. Engineering research especially with regard to reliability analysis of nuclear power stations **[134]**, aero space systems **[135]** and electrical equipment **[136]** started much earlier and followed in many cases the concept of fault tree analysis **[137]**.

On the other side, artificial intelligence offered new methods for treatment of cause-effect relations **[138]**, and for diagnostic problem solving **[139]**.

The development in the area of artificial intelligence was oriented initially to medical diagnosis and then extended to technical process. Therefore, also for technical diagnosis only heuristic symptoms were considered. Then more sophisticated diagnostic reasoning strategies were developed by increasing the level of abstraction **[140][141]** using the causalities as deep logic interdependencies. The fault-symptom trees known from engineering and AI strategies for treating causalities can be brought together for fault diagnosis. Especially the analytical symptoms generated by the model-based fault detection then allow performing a deep fault diagnosis by pinpointing on the possible physical fault origin.

3.3.2. FDD in the field of Ambient Intelligence

As showed previously, Ambient Intelligence is a field that has many applications, which vary from enhancing everyday life tasks to guaranteeing patients safety in hospitals. Such systems' services, which goal is generally to satisfy the user's preferences by performing a specific task, are executed in the background in a way that they are unnoticeable by the user. These characteristics cause some difficult challenges in fault detecting and diagnosing. Indeed, it is unacceptable for a system defined as **non intrusive** to flood the user with a large number of fault detection data. Conversely, not informing the users of detected faults may cause that users continue to rely on failed services without noticing **[1]**. A different approach is proposed in **[142]** to tackle this challenge by granting the ambient technology in this context the task of assisting the health carers via alerts, anticipating problems, explaining directives, etc. The technique is based on using cameras, equipped with night vision for overnight watches, and analyzing cameras feeds by specific algorithms that use causality and spatio-temporal reasoning.

Despite this, the challenge of guaranteeing the proper functioning of smart services is becoming even harder especially that Ambient Intelligent and Pervasive systems are becoming increasingly autonomous and complex, which makes diagnosis not only a nontrivial task but a very critical task as well, and in some cases, a matter of life or death for the user (Ambient Intelligent health monitoring systems).

A good fault detection and diagnosis system in this context should be dynamic and flexible enough to cope with the openness and the constant changes in system state, and, at the same time, reliable and precise enough to detect malfunctions with the best accuracy, in accordance with the context of use. Our motivation in this thesis is to provide a framework for Ambient Intelligent systems that shapes the system models in a way that makes fault detection and diagnosis process an intrinsic part of the framework and thus an easier task to perform, which enforce the self-diagnosis characteristic of the Ambient Intelligent environment, thus bringing it closer to being self-healing capable.

To do so, we investigate in this section some of the existing works that have been done in the Ambient Intelligent context for ensuring the creation of fault tolerant systems, such as fault detection, fault diagnosis, automatic healing, etc. Indeed context-aware and adaptive systems are systems that are constantly monitoring their surroundings (via sensors) and reacting to their environment (via actuators). Typically such systems have a context awareness middleware, usually composed of a context manager (continuously collecting context information) and an adaptation manager (responsible for applying a set of rules "adaptation rules" defining adaptive actions to take as context information provided by the context manager change) **[143]**. In such a context a fault is the triggering of an incorrect rule or the failing to trigger the correct rule. Detecting adaptation faults is challenging due to the fact that they are based on shared variables obliging to handle concurrent triggering of rules, their ordering, etc. Moreover, these context variables are

updated asynchronously according to different frequencies by the context awareness middleware, which leads to inconsistencies between the actual context and its representation within the system. These challenges were discussed in **[144]**; the proposed approach is based on the definition of a formal finite-state model of adaptation rules. Algorithms are then proposed to analyze the finite-state model in order to detect adaptation faults. In the model the states represent equivalence classes of stable configurations of context values, whereas the transitions represent the satisfaction of rule predicates. Thus a satisfied rule triggers an adaptive change from one state to another and can also trigger the execution of other associated actions. The limitation of this approach is the comparison of system state to an already established set of adaptation fault patterns, whereas, in this thesis, we dynamically deduce faults by comparing the real world state to the model dynamically at run-time.

Contrary to the latter approach, which is based on the assumption that context inputs in a context-aware application are correct and check whether there are faults in the triggering of the corresponding rules, the approach in **[145][146]** does not adopt this assumption, and so it performs inconsistency detection by identifying conflicts in context inputs at run-time before these inputs are fed into an application.

A different approach is introduced in **[31]**, in which a set of context ontologies define the possible run-time contexts. These contexts might be the devices' run-time statuses and the relationships between entities involved in service calls. The goal of the approach is to enable real-time self-management in an Ambient Intelligent environment. To do so the context ontologies are reasoned over by a set of rules that detects and handles specific events such as changes in states of devices, selection of the appropriate services according to the current context and the detection of malfunctions. Such an approach supposes that we can have a list of possible states for the Ambient Intelligent system devices. However, among the set of ontologies used in this approach is the "device malfunction ontology", which defines a hierarchy of possible malfunctions. For example the category "error" includes the "device totally down" state, which has a sub-category "battery error". There are also the concepts of "Cause" and "Remedy", which facilitate the self-healing task. We reckon that there are some unpredictable malfunctions that can occur in the Ambient Intelligent environment and that are caused by external factors. For example a lamp that is accidentally covered by another object will not emit light, even though according to the system state everything is working properly.

Other systems have been built specifically as an infrastructure for pervasive computing, such as:

- The Context Toolkit **[147]**, which is composed of context widgets that mediate between the environment and the Ambient Intelligent system to notify it of components and people presence, identities and current activities, allowing the system to be aware of the context and facilitating the development of context-enabled applications.

- Aura **[148]**, which is an architectural framework for handling the problem of resource variability caused by users' mobility in a ubiquitous computing setting. The approach is based on the use of user tasks models to configure, supervise, and adapt the Ambient Intelligent system resources in a proactive manner.

- Solar **[149]** is an implementation of a graph-based abstraction allowing the management of context information. The approach allows the sending of context information to the subscribed applications in the form of events flowing through a directed acyclic graph of event-processing operators.

- ConFab **[150]**, which is a toolkit providing a customizable framework for managing (including sharing) personal information in a privacy-sensitive way, allowing the construction of privacy-sensitive ubiquitous computer-based applications.

- Gaia **[151]** is an experimental middleware infrastructure that manages the Ambient Intelligent environment resources, detects and recognizes different contexts, and helps in developing and executing applications. The solution provides user-oriented interfaces and network-enabled computing resources allowing users to easily configure the system's behavior and to seamlessly integrate personal devices in the environment.

These infrastructures provide system-level mechanisms for monitoring application components and address particular issues that arise in Ambient Intelligence contexts such as context-aware management of heterogeneous resources and distributed computing. In addition to that, Gaia incorporates some fault-tolerance mechanisms. Moreover it has been extended with some fault handling techniques in **[152]**. These mechanisms include heart-beat-based status monitoring, redundant provisioning of alternate services/ applications, and restarting failed application components. Also different failure reasons were discussed and classified in the latter work. Indeed components and/or services may fail because of various reasons including low battery power, physical damage, network disconnections, etc. In the same context other techniques were presented such as **[153]**, in which a recovery model has been developed for context-aware environments. However the latter supports object-level monitoring mechanisms to detect object binding failures and focuses on application-level programmed error recovery mechanisms rather than on system-level mechanisms for building fault-tolerant and robust context-aware applications. In fault detection techniques that are based on comparing readings of each sensor with the model's calculated values, an inconsistency between the actual sensor reading and the modeled sensor value does not necessary mean a faulty sensor, it is after further analysis of the model that conclusions can be made on the fault cause.

Some approaches were presented for monitoring and diagnosis of real-time embedded systems, among which we cite **[154]**; it presents a framework for modeling faults in hybrid systems. Hybrid systems (systems that can behaves in both continuous and discrete manner) here are used to describe the continuous and discrete dynamics and interactions of sensor-rich embedded systems such as networked printers or automotive vehicles. The approach integrates model-based techniques using hybrid system models with distributed signature analysis. The modeling of fault is done using hybrid automata models. Faults considered are abrupt failures (discrete faults), which cause the increase of the computational cost, and subtle degradation of components (continuous faults), which are hard to be estimated efficiently. Using a simulation technique, the developed model is used to generate, off-line, a fault symptom table for different fault hypotheses. The table is then integrated into a decision tree used by the on-line diagnoser. **Figure 45** shows the architecture of the on-line diagnostic system for the example of the DC265 printer as an embedded system.

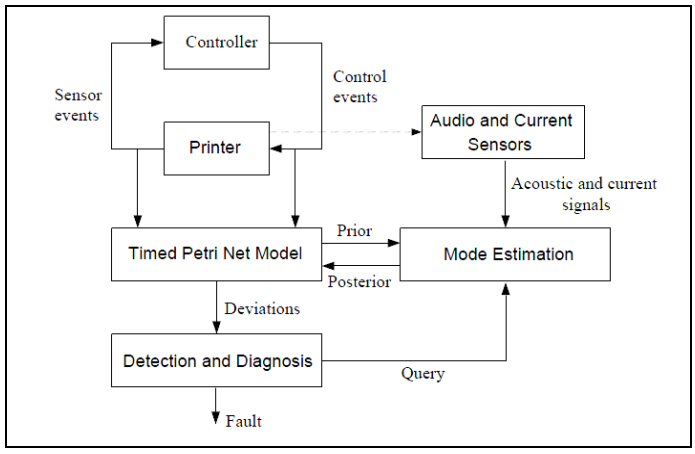

Figure 45. Architecture of the prototype diagnosis system [154]

In the on-line diagnosis system only the temporal discrete-event behavior of the hybrid system are simulated using a timed Petri net model, whereas the continuous dynamics are abstracted away. The model compares observed sensor events with their expected values. When a fault is detected the decision-tree diagnoser is triggered. The originality of the approach comes from the fact that it presents a fault parameterization that can model both abrupt failures and subtle degradations of components. Moreover, unlike existing approaches for monitoring and diagnosing hybrid systems, this approach address the computational data association issue associated with distributed multi-sensor systems by assuming that sensors' outputs have already been properly assembled to form likelihood functions of the system output. Online approaches are different from this approach since they do not model faults explicitly; instead they deduce their existence by comparing the sensors actual readings with the simulated ones.

Some diagnosis works have been done for networked embedded systems. The main idea is to distribute the amount of computation necessary to perform diagnosis among distributed nodes; each of which collaborates with others and communicates using wired or wireless network. The nodes are embedded in the environment using devices such as sensors and actuators.

A diagnosis system in such context may be centralized, decentralized or distributed.

Usually to perform centralized diagnosis in distributed systems, observations are made locally at node level; these observations are sent to a centralized diagnoser. The latter performs the diagnosis and re-distributes the results to the nodes. In other cases the centralized diagnoser only re-distributes orders based on the results of the diagnosis. However in networked embedded systems many constraints may not be met, for instance we may not have a processor with memory large enough to store a local image of the global diagnostic model and run the centralized diagnostic algorithm. Instead we have a small local processor in each node. This raises robustness and scalability issues that must be addressed. In some works, such as in **[155]**, the efficiency of a centralized diagnostic procedure for a distributed network of computers is increased using an approximate representation and carefully designed active probing of the distributed system.

To perform decentralized diagnosis a coordination process is used to allow the assembly of local diagnosers' diagnosis reports into a single global diagnosis **[156]**.

In distributed diagnosis architectures, each local diagnoser communicates directly with other diagnosers and the diagnosis task distributed over each local processor, and they become local diagnosers. To complete the diagnosis, every local diagnoser performs, locally, partial diagnosis, using a model of the locally concerned portion of the system and the locally available observations. Communication is then required to combine the local diagnosis reports into one global diagnosis. Based on this idea, a distributed diagnosis framework is presented in **[157]**. The framework exploits the topology of a physical system to be diagnosed to limit inter-diagnoser communication and compute diagnoses at anytime and using any information available, making it resilient to communication and processor failures. The framework adopts the consistency-based diagnosis formalism and develops a distributed constraint satisfaction realization of the diagnosis algorithm. The idea is that each local diagnoser first computes locally consistent diagnoses, taking into account local sensing information only. The local diagnosis sets are reduced to globally consistent diagnoses through direct communications between local diagnosers. This proposed approach aims to generate accurate diagnoses in a time-critical manner using the available computational resources. The challenges discussed in this work are mostly related to the nature of distributed diagnosis, such as Scalability, Robustness, Reconfigurability, etc. Whereas in our work we focus on overcoming challenges related to the specific nature of the Ambient Environments such as the dynamicity in adding and removing devices at run-time (openness) without impairing the diagnosis results. The Diagnosis is mainly concerned with verifying whether or not the actuators' actions were properly executed based on reading from sensors.

3.4. Ambient Intelligent System Modeling

Historically, diagnosis systems were based on if-then rules created by experts in a specific field. These rules are to be applied to systems that are from that specific field. The rules explicitly expressed the relation between the abnormal behavior and the possible faults in the system that might have caused it.

However, systems nowadays are becoming more and more complex, so such ad-hoc diagnosis systems became difficult to develop, and their maintenance and scalability became extremely difficult tasks. Moreover, it is almost impossible to provide a formal scientific measurement for the quality of such hand-crafted, and ad-hoc, diagnosis systems **[158]**.

Practically, in order to perform a rule-based diagnosis for a certain system, an expert should identify situations and states of the system to be diagnosed that might cause failures and then associate each state (cause) with the appropriate failure (fault) in the form of if-then rules.

Even though such rules are intuitive to make and easy to execute, they are very susceptible to conception errors and there is no formal way to verify the correctness of the rules, or to verify that all rules covering all possible faults have been created. In the case of model-based diagnosis system, we first start by creating a model describing the totality of the system (not only faulty states), and then the model is exploited by algorithms that reason to deduce the existence of faults and the component(s) that caused the fault.

Such a method guarantees scalability of the diagnosis system when the system model is updated **[158]**. This method also gives more flexibility for the modeling language to use to model the system. It is to be noted that the right diagnosis algorithms have to be chosen for the appropriate modeling language used. And executing these algorithms to perform on-line diagnosis task can be demanding in computational resources.

So in conclusion a fault-based diagnosis approach, regardless of the field or application in which it is used, can be summarized in the development of a model that simulates a real system and then apply diagnosis algorithms on the model. By doing so, we can observe and decide whether the behavior is normal or not (that is fault detection). When there is an abnormality, other algorithms are applied to isolate the system's component that is responsible for the abnormal behavior (that is fault diagnosis) **[159]**.

We reckon that model-based diagnosis approach are more adapted to complex systems, such as Ambient Intelligent systems, and that is why, in this thesis, we have adopted a model-based diagnosis approach.

In this part we expand more on the concept of a system model, which is a conceptual representation of a real system focusing on particular aspect of the system (structure, functioning, relationship with other systems, etc.). In the next part there are short definitions of the main types of models used in this dissertation. With these definitions we aim to clarify the difference between two particular types of models; the mathematical models, which are the models used in the field of automatic control (which is the field where the concept of fault detection and diagnosis of faults in a system was first introduced. "see **3.3**"), and the conceptual models, which are used to design most of software representations of real systems. In the literature, models describing systems, including conceptual models, are created according to two approaches: a non-architectural approach, where every aspect of the system (structure, behavior, data flow and communication between entities) is illustrated in a separate model, each of these models has its own syntax and particularities, and an architectural approach, where one single complete model incorporating every aspect of the system. In this dissertation we combine architectural (when we expose the FDD System in general) and non-architectural (when we zoom-in on a particular

aspect of our System to better detail it) approaches in order to give the most complete description of our Fault Detection and Diagnosis System.

- **Conceptual model**

In the field of computer science a conceptual model, as its name indicates, is a model that represents concepts and the relations between them. These concepts are also known as entities. The conceptual model is a high level representation of concepts and ideas in the sense that it is independent of implementation constraints **[160]**.

There are many notations to describe a conceptual model, among which we cite UML **[161]** and Entity Relationship Modeling **[162]**.

- **Mathematical model**

Mathematical models are theoretical descriptions of systems using mathematical equations. These models are used in a wide range of scientific fields in order to describe components or phenomena, and to analyze and/or predict actions (or their effects) of system's components **[163]**.

There are different types of mathematical models, among which we cite dynamical systems (in which time is considered) **[164]**, statistical systems (in which the mathematical equation relates variables according to stochastic and probabilistic theories) **[165]**, differential equations **[166]** **[167]**, etc.

- **Architectural model**

An architectural model not only describes the structure of a system, but also its behavior (general behavior and the behavior of the components) and the relations that the system has with its surrounding (for example other systems) **[168]**.

In this dissertation we use several kinds of models to describe our fault detection and diagnosis system, using heterogeneous modeling techniques:

- Block diagrams are used for the high-level description of components of our system and connections between them **[169]**.

- UML (Unified Modeling Language) is used for a more detailed description of the entities that make the components of the system.

- Finite State Machines (FSM) are used to describe the behavior of components. A (deterministic) Finite State Machine is a structure composed of states and transitions between states. At the start, the system under study is in a so-called "initial state", and at any moment, it is in exactly one of the states. When an event occurs in input, it can change state by following one of the current state's outgoing transitions. Therefore, transitions are labeled with "firing" events. A variant of FSM is called TFSM for Timed FSM. In such a state machine, some transitions are labeled not with an input event, but with a duration: these are called *timed transitions*. When entering a state, a timer is reset. If at some point it reaches the duration of an outgoing timed transition, then the transition is activated. If the state changes before the timer elapses, the timer is deactivated.

- **Multi-paradigm modeling**

As seen above, when modeling a complex system such as an Ambient Intelligent system, one has to deal with several kinds of models. For instance, if we want to build a model to predict the expected output of a light sensor in a room with a window, we must combine:

- A block diagram relating physical variables, consisting of mathematical operators,

- A finite state machine modeling the state and state changes of objects, such as the automatic blinds of the window.

Therefore, the overall model of the environment is heterogeneous: it combines models built using several modeling languages, several *modeling paradigms*. The field of multi-paradigm modeling **[170]** studies the ways in which these modeling paradigms may be combined, how the semantics of the resulting model can be defined precisely.

Defining the meaning of heterogeneous is difficult **[171]**, first because it requires to precisely specify the semantics of each one of the modeling languages used in the model, and second because it requires to define how information is interpreted at the boundaries between the models of the heterogeneous subsystems. Indeed, different modeling paradigms may use different structures for data (arrays, samples, functions), different notions of time (continuous, discrete, periodic, triggering), and different ways of combining the behavior of the elements of a model (sequential, concurrent, synchronous, with blocking communications). These semantic components define the underlying *Model of Computation* (MoC) of the modeling paradigm **[172]**.

Well-known MoCs include (Timed) Finite State Machines (FSM, TFSM), Discrete Events (DE) or Synchronous Data Flow (SDF). We have described FSM and TFSM above. Both SDF and DE fall in the category of "block diagrams".

SDF **[173]** enables one to model sampled systems: a model is a graph composed of operators that exchange flows of data tokens at fixed rates. An SDF model may be analyzed and scheduled statically using a simple mathematical method.

DE **[174]** enables to model sub-systems in which components exchange messages at specific instants. For instance it is well suited to model the exchange of messages on a bus or on a network. Each message has a value and a time-stamp, and if several messages have the same time-stamp, they are delivered in a sequence of microsteps (determined by a topological ordering of the components), so that the overall observation at that time is causal and deterministic.

Several categories of approaches, like model transformation, language composition or model composition, address the problem of modeling a system composed of heterogeneous components. A classification and a comparison of these approaches is proposed in **[171]**. Among them we can cite Ptolemy II **[174]**, Metropolis **[175]** the MATLAB/Simulink toolchain by The MathWorks, and ModHel'X **[172]**.

Ptolemy II **[174]** is one of the first approaches for model composition. It supports a wide range of MoCs that may be combined with each other to form heterogeneous models. The adaptation between MoCs is hard-coded and cannot be changed. If the model designer wants to use his/her own adaptation scheme, he/she must explicitly add adaptation blocks into the models themselves. Such artifacts render models quite difficult to understand and to reuse.

Metropolis **[175]** is a heterogeneous system design environment which relies on the separation of communication and computation concerns. Metropolis models are made of communicating processes. Heterogeneous processes may be connected through *adapters*.

MATLAB/Simulink supports a set of hardcoded MoCs, for instance a variant of TFSM (Stateflow) and a variant of SDF (Simulink). The semantic adaptation between a Simulink and a Stateflow models can be specified explicitly using functions and truth tables. However, all MoCs cannot be composed in the same way. For instance, using a Simulink (SDF-like) model into a

SimEvents **[176]** (DE-like) model requires different adaptation artifacts such as event translation blocks **[175]**. Therefore adaptation is ad-hoc, specific to every pair of MoCs.

To address these issues, a new approach called ModHel'X is being developed at Supélec **[172]**. ModHel'X allows the composition of heterogeneous parts through hierarchy: at a given level, a model is homogeneous, but it can contain blocks that embed other models that obey different models of computation. These special adapter blocks are called *interface blocks*. ModHel'X introduces abstract semantics that allow the easy addition of new MoCs and new types interface blocks. ModHel'X will be used in the implementation of our approach, so it is described more precisely in Section 5.2.

3.4.1. A general architecture for a typical Ambient Intelligent Environment

From the above definitions we can draw a very general architecture of a typical Ambient Environment (see **Figure 46**).

As illustrated, the architecture is user-centered. An Ambient Intelligent Environment (Smart home, smart hospital, smart factory, etc.) is equipped with an Ambient Intelligent System. The Ambient Intelligent System has an operational layer (hardware) and an intelligent layer. The user can interact with the Ambient Intelligent System that is installed in his/her surrounding, either directly via the available human computer interfaces available, or indirectly by acting upon and changing the Ambient Intelligent Environment. In the latter case scenario, the Ambient Intelligent system can detect these changes via sensors. The Ambient Intelligent System can also act upon the surroundings in order to change their state via actuators.

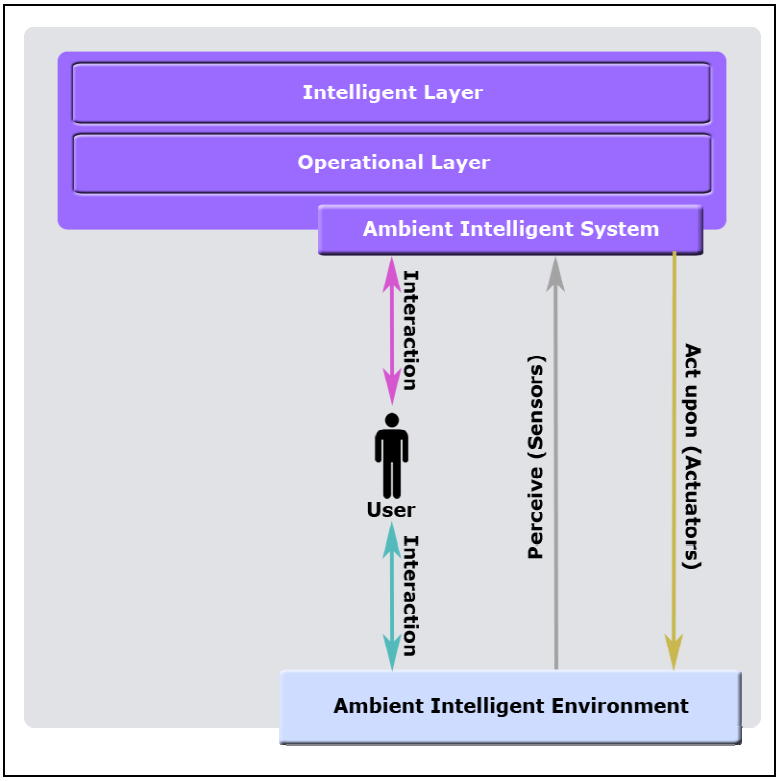

Figure 46. A typical Ambient Intelligent Environment

In this thesis we focus on the "act upon"-"perceive" relationships between the Ambient Intelligent system and the Ambient Intelligent Environment, as we aim to add a Fault Detection

and Diagnosis (FDD) layer to monitor and supervise system tasks. The FDD layer proceeds to simulate the "act upon"-"perceive" actions of actuators and sensors in order to predict sensors readings and compare them with actual readings to detect anomalies.

3.5. Conclusion

In this chapter we gave an overview of the field of Ambient Intelligence and we cited some application examples of Ambient Intelligent Systems. Then we gave some definitions from the field of fault detection and diagnosis, in which we introduced the classical approach and cited some fault detection and diagnosis approaches applied to the field of Ambient Intelligence.

In our work we adopted basic definitions from Fault Detection and Diagnosis field (such as differentiation between the Fault Detection task and the Fault Diagnosis task). We noticed that even though some of these techniques were applied to Ambient Intelligent Systems, they were not fully adapted to handle the particularities of Dynamicity and Openness of such Systems. In this thesis we give a new method for verifying the proper conduct of Ambient Intelligent System actions via a generic modeling of Actuators (that are performing the actions), using information from available resources (mainly sensors' readings), via a (more or less) detailed definitions of the physical phenomena that are observed in the Ambient Environment. These physical phenomena definitions allow decoupling Actuators and Sensor at design time (since we can deduce these links at run-time by applying the definitions to actual devices), thus overcoming Dynamicity and Openness of Ambient Intelligence when designing a Fault Detection and Diagnosis System. In the next chapter we present our approach, and we detail our Fault Detection and Diagnosis framework that implements the proposed approach.

Chapter 4:

AmILoop: A Fault Detection and Diagnosis Framework for Ambient Intelligence

Chapter 4.

AmILoop: A Fault Detection and Diagnosis Framework for Ambient Intelligence

In this section we detail our Fault Detection and Diagnosis framework for Ambient Intelligent Environments. We start by presenting our Fault Detection and Diagnosis general architecture. In this part we examine the different models constituting the framework and the relationships between them. Then we examine our models' composing entities a little deeper by detailing their structure and their role in the fault detection and diagnosis tasks, mainly we define our new concept introduced in this framework.

In **Figure 47**, we can see the FDD framework situated within the context of a real Ambient Intelligent System. The latter's most important components, that are necessary to the operation of the FDD framework, can be classified in two main types that are actuators and sensors. Such components, when used in the context of an Ambient Intelligent Environment can be dynamically added or removed at run-time. That is why we cannot have classic ad-hoc control loops to control the proper functioning of the Ambient Intelligent System. The FDD framework introduces new entities and techniques that permit to deduce the links between actuators and sensors at run-time.

These components (and other entities), how they communicate, and how their data is being used by our framework are described in this section. We describe the structure of our framework by constructing a hierarchy of models and we simulate the use of these models, at run-time, when performing Fault Detection and Diagnosis Tasks.

As illustrated in **Figure 47**, to construct the models that describe the Ambient Environment, the FDD framework relies mainly on an abstract model of the environment. From this abstract model an environment concrete model is deduced. The latter is finally instantiated to represent real components of the ambient environment.

- The *abstract model of the environment* is detailed in **Figure 48**. It defines the structure of the environment model in a way that enforces the decoupling of sensors and actuators at all levels. This is achieved by introducing the concept of **Effect**, which is a modeling of the physical consequence(s) of the actions of actuators onto the environment. The Abstract Model and the concept of effect are further discussed in **Subsection 4.1.3**.

- The *concrete model of the environment* follows the general structure of the abstract model and defines sensor and actuator types, the expected physical effects, the appropriate physical laws and the relations between all these entities.

- The *environment instances* are created at runtime by the context engine that intercepts system events and signals. It contains the actual sensors and actuators as well as the actual values of effects produced by actuators and read by sensors.

Because the models of a particular Ambient Intelligent system follows a common abstract model, it is exploitable by the *prediction engine*, responsible for deducing the values expected to be read by the sensors. Comparing these values with the actual sensor readings makes it possible to

perform Fault Detection. Then, using a diagnosis model, the diagnosis engine is responsible for isolating these faults and determining exactly what components are responsible for them. In the following subsections, we discuss the different models composing our FDD framework and the fault detection task.

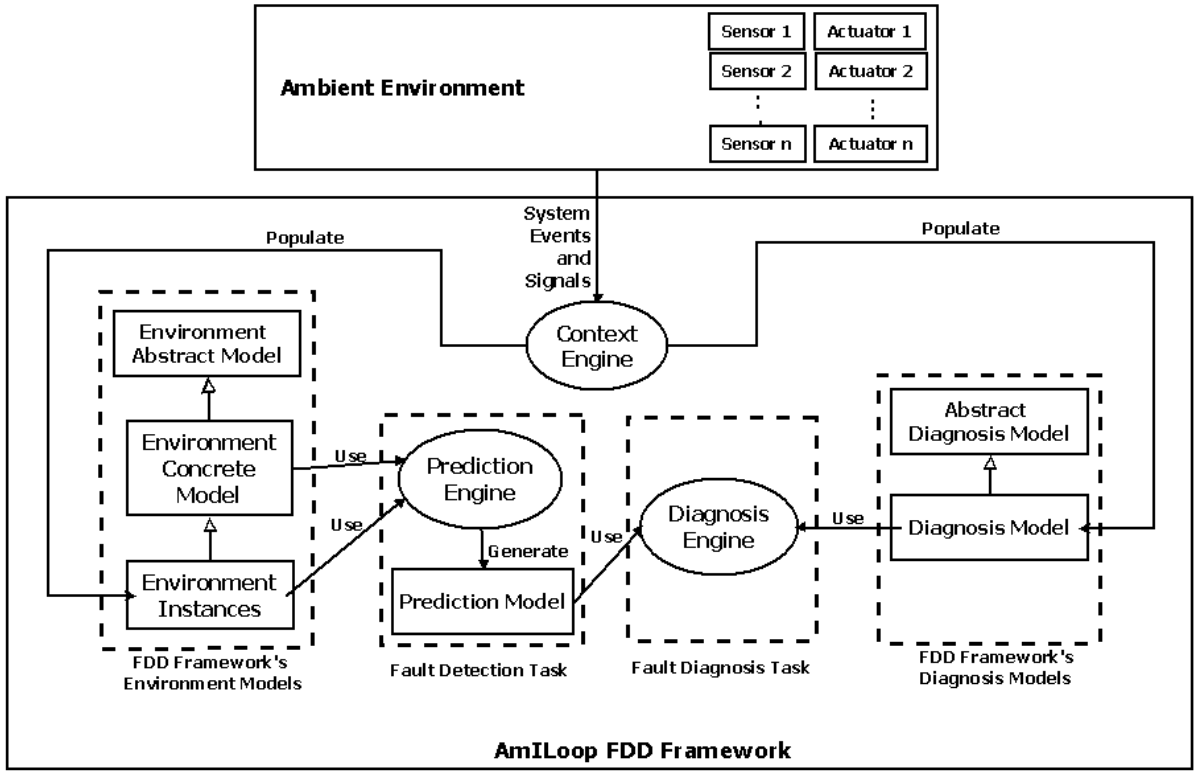

Figure 47. The FDD Framework Architecture in the AmI context

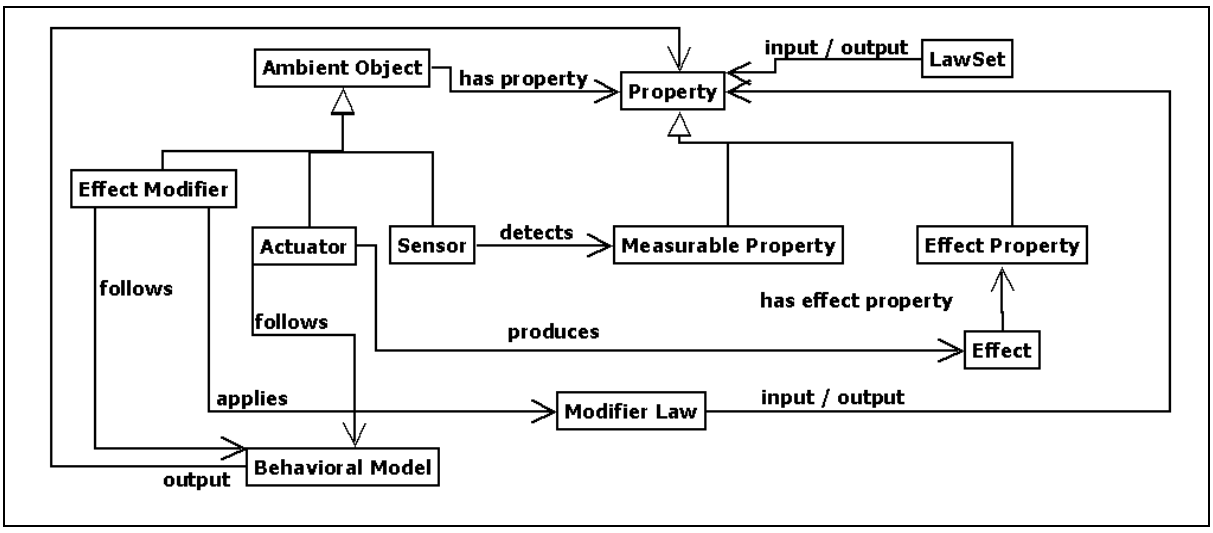

Figure 48. FDD Abstract Model

4.1. General Architecture of the FDD Framework

To better explain our approach we will look at the FDD framework from two perspectives;

(i) An architectural point of view: A general view for the framework's overall structure, hierarchy, and operations of the FDD framework (see **Figure 51**), in particular how the constituting models are generated, populated (with new types or instances), updated, and/or exploited by the FDD framework's engines to perform the fault detection task.

(ii) A conceptual point of view: A more detailed look into the FDD Framework's Models (see **Figure 49** and **Figure 50**), which describes the way the ambient environment and its components are modeled within the framework.

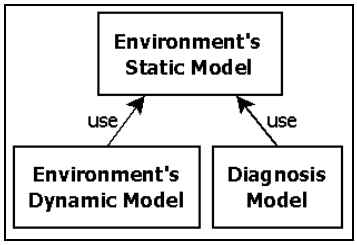

Figure 49. The FDD Framework Models

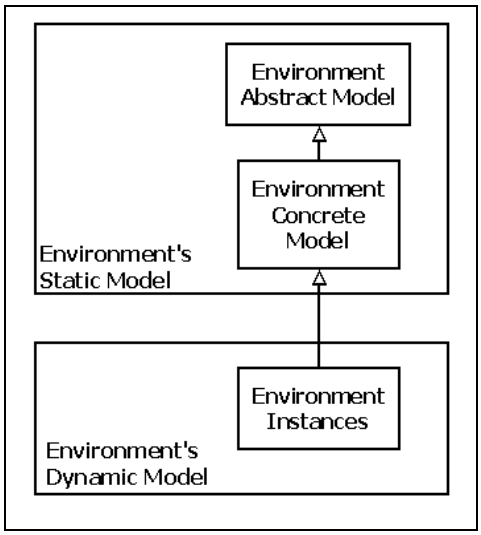

Figure 50. The FDD Framework Models' Hierarchy

4.1.1. Architecture of the FDD framework: from a general structural point of view

In order to the FDD Framework to perform the Fault Detection and Diagnosis tasks it uses information from the following models:

- The Environment's Static Models: defined and updated by the designer of the system
- The Environment's Dynamic Model: contains the instances corresponding to the actual devices in the environment. It can be updated by the system at run-time in order to add/remove/update devices or values associated to the devices.
- The Diagnosis Model: contains information that can be used to pinpoint the source of the detected fault.

In **Figure 47**, the "use" relation between the models describes in fact the way the FDD framework uses information from one model to construct the other. For example, as explained earlier, the FDD framework uses information from the static model in order to populate the dynamic model (create environment instances).

The Diagnosis Model is used to achieve fault isolation. Therefore its nature is completely dependent on the type of Diagnosis Engine used. We neither restrict the range of fault isolation techniques, nor the nature of the diagnosis model to be used. In all cases however, the FDD framework uses the static model to build or complete the Diagnosis Model.

4.1.2. Architecture of the FDD framework: a behavioral point of view

The operations of fault detection and fault diagnosis depend on specific models from the FDD Framework. These models are exploited by engines in order to deduce fault detection and diagnosis conclusions. The general run-time behavior of the framework, as shown in **Figure 51**, can be summarized by these steps:

iv) The **Context Engine** uses information from the hardware layer and from the Environment's **Static Models** (composed of the **Abstract Model** and the **Concrete Model**) to properly instantiate the real-world objects and initializes their attributes values (actual positions, actual readings, current state, current emitted value, etc.).

v) Information contained in the Environment's **Static Model** (Physical Laws to apply and/or Deduced Links between Different Types of Actuators and Sensors) and information contained in the **Dynamic model** (Actual Instances and their values) are used by the **Prediction Engine** to generate the **Prediction Model**. The latter is a combination of Behavioral Models and Mathematical Models allowing the calculation of the expected values of sensors' readings. The comparison between predicted values and real readings of sensors allows us to detect probable faults. This is the **Fault Detection** task.

vi) These conclusions (probable faults) with the calculated values for sensors readings, information about the current System structure, component states, links deduced at run-time between components, etc. which exist in the **Prediction Model**, and information from the **Diagnosis Model** are exploited by the **Diagnosis Engine** to perform **Fault Isolation**. The **Fault Diagnosis** task is then complete.

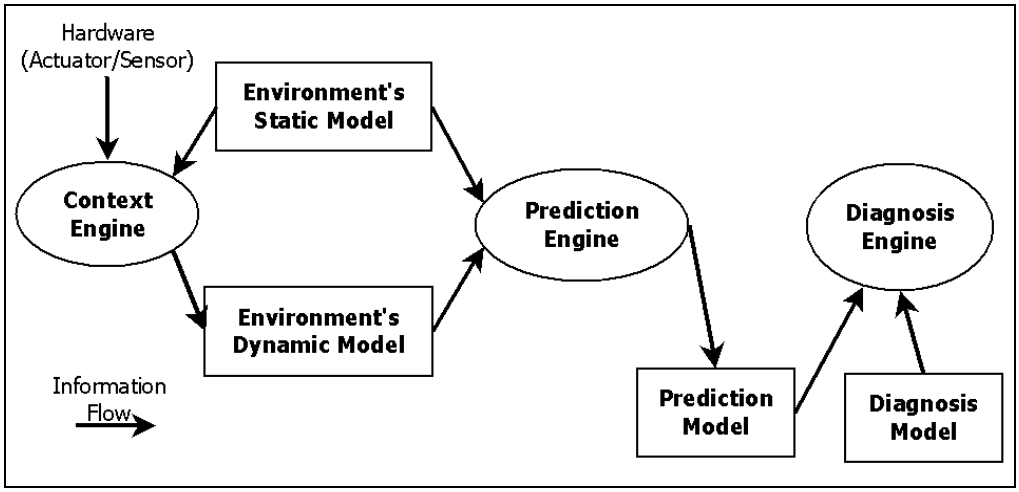

Figure 51. Run-time Architecture of the FDD Framework

4.1.3. The FDD Framework Models: actuator-sensor decoupling and main concepts

4.1.3.1. Actuator-sensor decoupling

In this part we show how the abstract model allows one to model the ambient environment while enforcing the decoupling of actuators and sensors at design time. As a matter of fact, one of the challenges for creating classic control loops in an Ambient Environment is the dynamicity of adding and removing devices at run-time. Such property makes impossible to predict such control loops at design time. Our framework overcomes the necessity of coupling actuators and sensors at design phase by introducing the concept of effect. The latter, with help of mathematical formulas representing the physical phenomena observed in the environment, deduces links between components in the environment at run-time. The dynamic deduction of links between actuators and sensors is further detailed in **4.2**.

The general idea around actuator-sensor decoupling is illustrated in **Figure 52**:

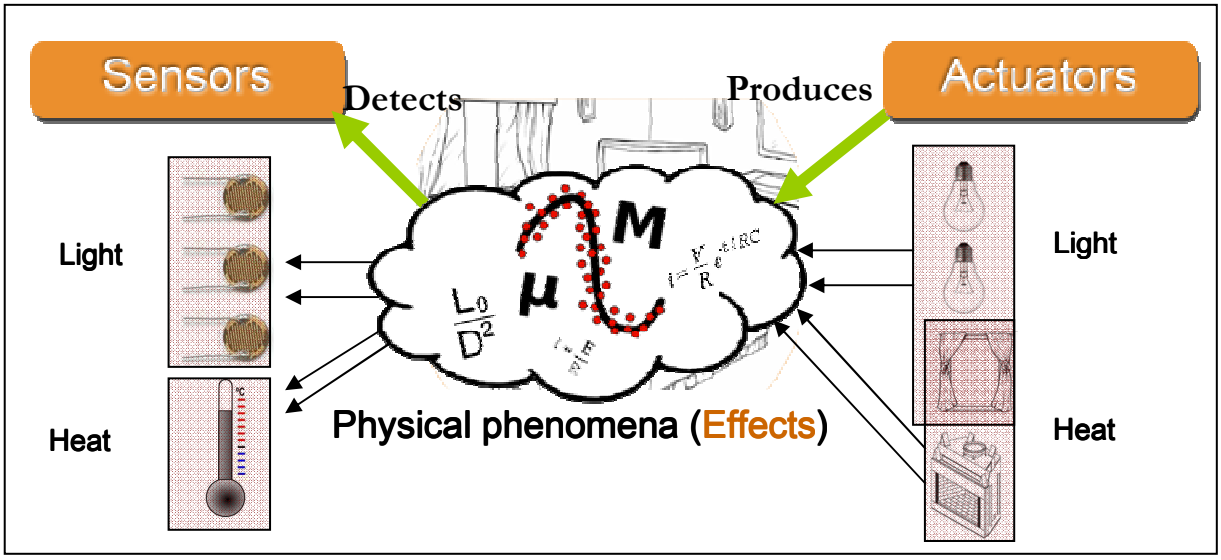

Figure 52. Simplified example of actuator-sensor decoupling

The idea is define actuators as "producers" of physical effects and sensors as "detectors" of physical effects. The mathematical formulas will automatically couple each sensor with the corresponding actuator(s). In addition to the decoupling, such definition allows a design of an Ambient Environment that is closer to reality since we can consider special cases where an actuator is producing more than one physical phenomenon in the environment. For a certain actuator, this can be by design (A window affects the light and the temperature of the room) or an accidental secondary effect (A fireplace can affect the ambient light of a room).

As we can see in the Abstract Model in **Figure 48**, we impose a certain structure to the model that will inherit from it (the concrete model). This structure allows, by forcing the definition of effects, the decoupling of Actuators and Sensors. This permits the modeling of the ambient environment without knowing in advance what the types of the components of the Ambient Intelligent system are, which means in consequence, modeling the environment without knowing in advance what the relations between the objects (mainly actuators and sensors) in the environment are going to be; this is very suited for the openness of Ambient Intelligent systems. What allows the deduction, at run-time, of these relationships is the key element of our model, which is the effect and the physical laws that will exploit the effect properties.

4.1.3.2. **The FDD framework models and main concepts**

As shown in **Figure 50**, the environment model can be divided into two main parts: a static one and a dynamic one.

The static model contains (i) *the abstract model* composed of generic entities, namely Actuator, Sensor, Effect, etc. and (ii) *the concrete model* that specializes and concretizes these entities (Light Sensor, Sound Actuator, etc.). Actuators produce Effects, which have Effect Properties (**Figure 48**). Sensors detect Measurable Properties. Laws relate all these kinds of Properties in order to model physical phenomena. Using laws it is possible to estimate the values detected by the Sensors.

The dynamic model contains the actual instances of sensors and actuators located in the physical environment. It stores the current state of the environment (sensor values, actuator commands) and it is updated continuously at run-time.

It is to be noted that even though the concrete model is considered as part of the static layer, it can be updated at run-time in the case where a new device (instance of actuator, or sensor, or modifier) that is of a new type is introduced. In that case we might have to add, in addition to the new device type, a new type of effect, effect property, measurable property, and/or law set.

Let us see how this works on a concrete environment model, corresponding to a lighting system. The abstract Sensor entity is concretized as a Light Sensor entity (or a specific Light Sensor Type), the abstract Actuator entity as a Light Bulb (or a specific Light Bulb Type). Light Sensors and Light Bulbs share an Ambient Property which is the Zone in which they are located (for example the name of the room). A Light Sensor can detect a light level (Ambient Light concretizes Measurable Property). Likewise, a Light Bulb produces a Light Effect (concretization of Effect) which is characterized by a Light Intensity (concretization of Effect Property). A corresponding set of Laws is instantiated in order to calculate the value of the Light Intensity around the Light Sensor entity.

The calculations will use properties such as the position of the Light Bulb and the Light Sensor to determine the distance between the two components, the light intensity emitted by the Light Bulb to determine the received light intensity. A combination law can be used if there is more than one Light Bulb emitting light toward the Light Sensor. It is important to note here that our approach does not impose a level of detail for the physical laws. It is up to the designer to choose the relevant level of granularity. Indeed one can imagine a different modeling for our example, in which the effect of light is represented by a Boolean value (light absent – light present). This freedom to choose the level of granularity is well adapted to Ambient Intelligent systems since their use in real world varies according to context. We can imagine a smart home design for people with hearing impairment in which the modeling of the effect of sounds is very detailed in order to enhance the perception of sound.

In the rest of this chapter we will focus on some key entities of the *Abstract Model*, we will explain their role, their relationships, etc. and we will show how we construct the *Concrete Model*'s entities by instantiating these abstract entities, thus creating actual types that are going to describe actual devices and physical phenomena that are observed in the Ambient Intelligent Environment.

4.2. The Concept of Effect

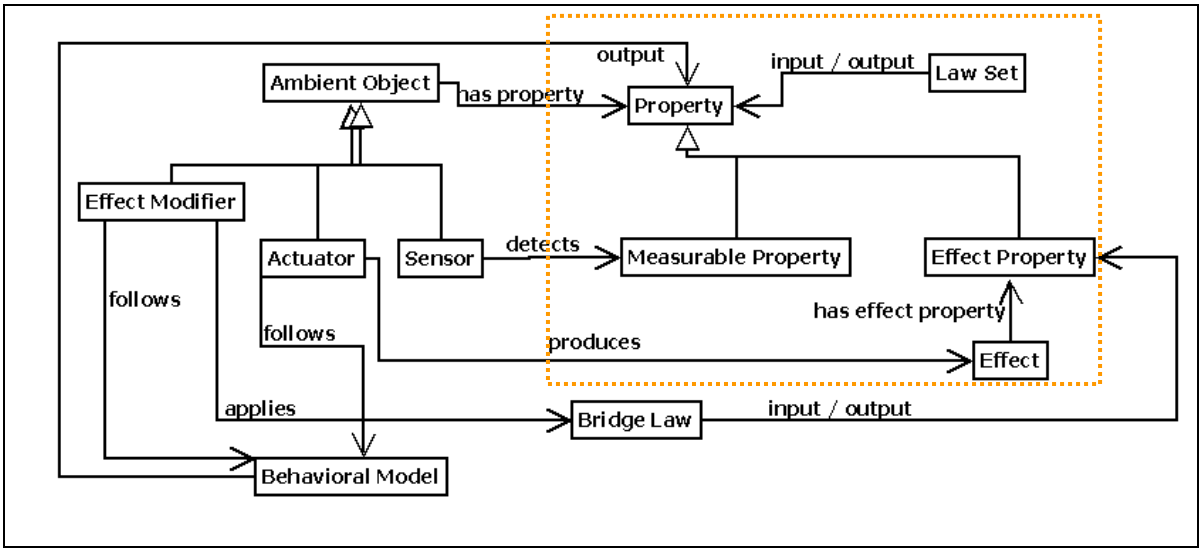

Figure 53. The abstract model entities modeling "effect" and "law set"

In **Figure 53** we see the entities that are involved in defining the concept of effect (not to be confused with the entity *Effect*) which are: *Effect, Effect Property, Law Set, Property and Measurable Property*.

The purpose of the concept of effect is to enable the simulation of the physical consequences of an action in an Ambient Intelligent Environment. In the proposed approach the concept of effect becomes the complete definition of a specific physical phenomenon that can be observed in the environment. The concept of effect plays the role of the (only) link between the actuators and the sensors; thus making it possible to model an Ambient Intelligent Environment while decoupling actuators and sensors at design time.

As a matter of fact, in an ambient environment, actuators, when receiving orders sent by the system, emit actions whose actual effects on the surrounding environment are only visible to the system through its sensors. We consider these actions from the point of view of the physical environment; accordingly they are defined as one or more physical phenomena. The format, the value and the way these actions are perceived by the sensors and processed by the system follow corresponding *physical laws*. These laws depend on a number of well-known physical parameters. In the proposed approach an explicit list of the effects that are expected to be observed in the environment must be defined and modeled by the system designer. For each declared effect a number of properties are defined too. These properties, which we will call *effect properties* from now on, correspond to the physical parameters defining the physical laws mentioned before. If we observe carefully the nature of the data collected by sensors, we conclude that these effect properties match exactly what sensors are sensible to. Even though this list of properties can be very explicit, it is to be noted that in reality the number and the nature of the properties are conditioned by the hardware configuration (the nature of the sensors and their precision) and the degree of details wanted by the user. In the latter case, the degree of granularity can be chosen according to:

- the global context of use. For instance we can imagine the designer choosing a very detailed definition of the effect of sound in an Ambient Intelligent Environment designed for assisting users with hearing impairments.

- or the current context of use. For example we can imagine an automated scenario that, in order to detect light-related malfunctions, uses a detailed definition of the light effect (modeled

using classical laws of physics for light propagation **[177]**) by night; and then uses an undetailed definition of the same light effect (using a very simple On/Off rule) during the day, allowing only the detection of the existence (or not) of external light from windows and doors, thus deducing malfunctions that are related to the state of window stores. The same undetailed rule can also be used at night in special cases (a closet) to detect the state of a light bulb (on or off); the rule would say "if a light bulb is *on* in a room then the light sensors that are in that room should *detect light*".

Practically, our motivation for proposing such a definition is to be able to predict from the model, at run-time, the expected value a sensor is supposed to read. By comparing these theoretical values to the actual readings of a sensor, we can deduce inconsistencies around sensors. Hence we can conclude whether or not actuators (effect creator) that are connected to the "faulty" sensor, has completed their actions successfully.

As the definition of the effect can follow different levels of granularity, an actuator that produces more than one effect can have its different effects defined according to different levels of details. In fact, an effect may be defined more than once depending on the importance of the produced effect with respect to each device. For example the heating effect generated by light bulbs is not as significant as the heating effect generated by heaters; nevertheless it might be important to model the light bulb heating effect in a particular scenario, like in the case of modeling the effect of a strong light projector. We can push the model further by defining the same effect that is produced by the same device more than once with different levels of details, for instance when an actuator generates effects in different rooms. In this case we can have the same effect represented in different level of details in each room. The diagnosis results from the different levels can be useful for the system's overall diagnosis. This generality and flexibility of the definition allows us to have more or less realistic definitions of the physical laws depending on different criteria such as the architecture of the system, the diagnosis technique used for the system, how accurate we want the diagnosis result to be, the desired level of detail we want for the diagnosis report, the context of use of the ambient environment, etc. Such flexibility is well-suited to the dynamic and heterogeneous nature of ambient environments.

An Effect is either produced by an actuator, or it is conducted from one zone to the other by an effect modifier (ex. a window). Each effect type has a certain number of properties that are used by a number of mathematical functions modeling the physical laws; the goal is to estimate, using effect properties and other properties of the objects, the value of the reading of a sensor sensible to that specific effect.

We chose to group these mathematical functions in sets we call *law-sets*. A set of mathematical functions models the entire corresponding physical phenomenon according to a certain level of granularity. An effect can be exploited by one or several law sets. The choice of which law set to use depends on the available information at the time of use of these functions by the prediction engine. We can imagine a hierarchy of law sets to choose from according to the level of detail of the data available to the Ambient Intelligent system.

In our work we use the proposed fault detection and diagnosis framework to model, as examples, the following types of Effects: Light Effect, Heat Effect, and Water Flow Effect. Following the same modeling steps, other effects can be modeled according to different level of details.

4.2.1. The Light Effect

Having the 'right' ambient lighting is one of the requests of ambient environment users. Light as a physical phenomenon is any radiation that is capable of causing a visual sensation directly **[178]**. We call 'Light Effect' the light waves produced by light sources. It is characterized by

luminous flux (in lumen). The observed property by light sensors is ambient light intensity (in lux). See **Annex-A** for detailed definition. In the next sections we first describe the light phenomenon from a physical point of view, then from a modeling point of view; finally we describe the concept of "light effect" as it is presented in our framework.

4.2.1.1. **The Light Effect – the concept**

The light effect is the entity modeling the physical phenomenon of emitting a visible light. In the proposed model light effect can have different definitions according to the wanted level of details. The difference between the different definitions is the Light Effect Properties. For instance we can imagine light effect with one property "state" that describe whether or not there is light. In this case state would have the values On/Off (or 0/1). We can imagine a more detailed definition of the light effect in which we have the luminous flux value of the emitted light. In this case we would use standard optical measurement laws described in **[179][180]** (see **Annex-A** "Light as a physical phenomenon – definitions" for detailed physical definitions).

These laws will be integrated in the model using law sets (see next paragraph).

4.2.1.2. **Deduce Illuminance from luminous flux–mathematical modeling of physical laws**

The aim of modeling the light effect is to allow the calculations (prediction) of the theoretical value of the Illuminance, that is caused by a light source's known luminous flux, around a light sensor (or any point of the environment). To do so we define a certain number of laws with which we define a law-set. The law set is composed of mathematical functions that are used in real time to perform the calculations.

In input, the mathematical functions use the values of properties from instances and results from other functions within the same law set. It is to be noted here that one of the most important physical attributes to calculate in the case of light is the *distance* between the light source and the point at which we wish to calculate the Illuminance. To estimate this distance the positions of the concerned object is to be known. The description with wish the position is described can be organized according to its level of detail. For instance when the Ambient Intelligent Environment components positions are described according to a 3 dimensions coordinates system (x,y,z) we say that it is the most detailed. We can imagine a hierarchy of law sets from the most detailed to the least detailed. The most detailed being the 3D light law set, then the 2D light law set, where components' positions are described via (x,y) coordinates, and finally the least detailed law set where each component is defined to be located in a zone, in which only components having the same value for the property *zone* can interact with each other. The Ambient Intelligent system starts by trying to evaluate the most detailed law set, if the latter is impossible to apply (due to a lack of information in the model), the system tries the next least detailed set, and so on.

In output, the only constraint is to have only one mathematical function within the law set that calculates the measurable property meant to be measured by the sensor, in our case the ambient light Illuminance.

In the following mathematical formulas we use a pseudo-code syntax. Each mathematical function has an identifying name, and it has a specific number of arguments that represent the instance(s) on which we are performing the calculations. In the body of a function we can have: *standard mathematical operations* and/or *constants*, we can also use a *call to another function* by its name, and/or we can use the value of a *property of an instance* (an instance of an entity from the model) using the following syntax:

```
property(entity instance)
```

which gives the value of the *property* for the specified *instance* in argument. For example to have the value of the property *x* of the instance *s* of the entity *sensor* we use :

```
x(s)
```

Below is the hierarchy of ambient light law sets (each with its mathematical functions) from the most detailed law set to the least detailed law set. It is to be noted that the most detailed law set will give more accurate fault detection results than the least detailed law set.

4.2.1.2.1. *Functions in a 3 dimension described environment (most detailed)*

This is the most detailed law set for the light effect. In the model we call this law set the *3DLightLawSet*.

The *3DLightLawSet* is composed of the following functions:

$$\text{sameZone}(s,a) = zone(s) \equiv zone(a) \tag{L1}$$

$$\text{distance}(s,a) = \begin{cases} \sqrt{(x(s)-x(a))^2 + (y(s)-y(a))^2 + (z(s)-z(a))^2} & when\ \text{sameZone}(s,a)\ is\ true \\ +\infty \quad when\ \text{sameZone}(s,a)\ is\ false \end{cases} \tag{L2}$$

$$\text{directLightExposure}(s,a) = \frac{\text{luminousFlux}(a)}{\text{distance}(s,a)^2} \tag{L3}$$

$$\text{ambientLightIntensity}(s) = \sum_{a \in LightEmitters} \text{directLightExposure}(s,a) \tag{L4}$$

In the previous law set

(L1) verifies whether or not s and a (generally a sensor and an actuator) are in the same zone. The '≡' operator is the equality operator. After converting zones values to integers, this function can be implemented mathematically with the function: $0^{(\text{zone(s)-zone(a)})}$. So it returns 1 when a and s are in the same zone, and 0 otherwise.

(L2) uses the x,y coordinates to calculate the distance between an actuator and a sensor that are in the same zone. This function returns an infinite distance value when the two objects are not in the same room. We can imagine that this function divides the results of the square roots (in all cases) with the result of (L1) for the same objects s and a.

(L3) estimates the light intensity value at a light sensor when exposed to a single light source positioned at a certain distance (calculated from **(L2)**) and generating a certain luminous flux. The input parameter luminous flux is the effect property that ensures that **(L3)**, and consequently the whole ambient light law set, only considers actuators that produce a light effect.

(L4) calculates the sum of all the results from **(L3)**, which is the sum of the light intensities caused by each single light source on this particular light sensor. **(L4)** is the function that calculates the theoretical value of the measurable property ambient light intensity (Illuminance) around a sensor. The comparison of this value with the actual reading of the sensor is the basis of the fault detection task.

Note here that Light Emitters does not necessary mean only Light Actuators (in **4.5** we will introduce the concept of Effect Modifier that can also play the role of Effect Emitter in general).

In **Figure 54** we see the call tree for the *3DLightLawSet*. This call tree gives us an idea about how the prediction engine part of the Fault Detection and Diagnosis framework (see **Figure 47**)

construct the prediction model. In fact every instance of this call tree (instantiated for sensors that have as the measurement property, the output of the law-set) is a prediction model for a certain sensor. The instances of prediction models constitute the System's Prediction Model. Every sensor's prediction model evaluates the law set and predicts the value of the reading of the corresponding sensor. A successful evaluation is an evaluation that covers all the leaves of the call tree. The leaves of the tree are represented with ellipses, whilst the rest of the nodes are represented with rectangles.

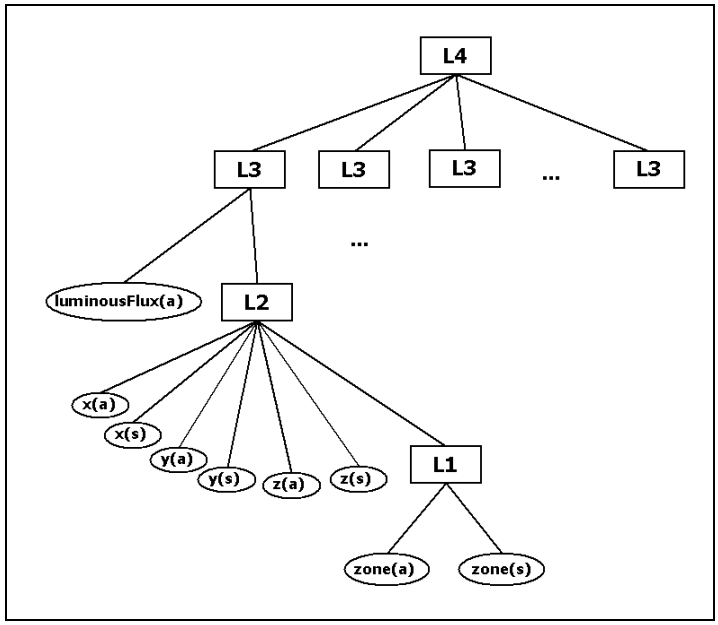

Figure 54. Call tree for the 3D Light Law Set

4.2.1.2.2.***In a 2 dimension described environment (less detailed)***

With this law set we go down one level in the hierarchy of granularity of the law sets defining the light effect. In the model we call this law set the *2DLightLawSet*.

The *2DLightLawSet* is composed of the following functions (note that only the distance function **(L2)** from the previous law set is replaced by **(L5)**):

$$\text{sameZone}(s,a) = zone(s) \equiv zone(a) \tag{L1}$$

$$\text{distance}(s,a) = \begin{cases} \sqrt{(x(s)-x(a))^2 + (y(s)-y(a))^2} & when\ \text{sameZone}(s,a)\ is\ true \\ +\infty \quad when\ \text{sameZone}(s,a)\ is\ false \end{cases} \tag{L5}$$

$$\text{directLightExposure}(s,a) = \frac{\text{luminousFlux}(a)}{\text{distance}(s,a)^2} \tag{L3}$$

$$\text{ambientLightIntensity}(s) = \sum_{a \in LightEmitters} \text{directLightExposure}(s,a) \tag{L4}$$

The call tree for the 2D Light Law Set is depicted on **Figure 55**

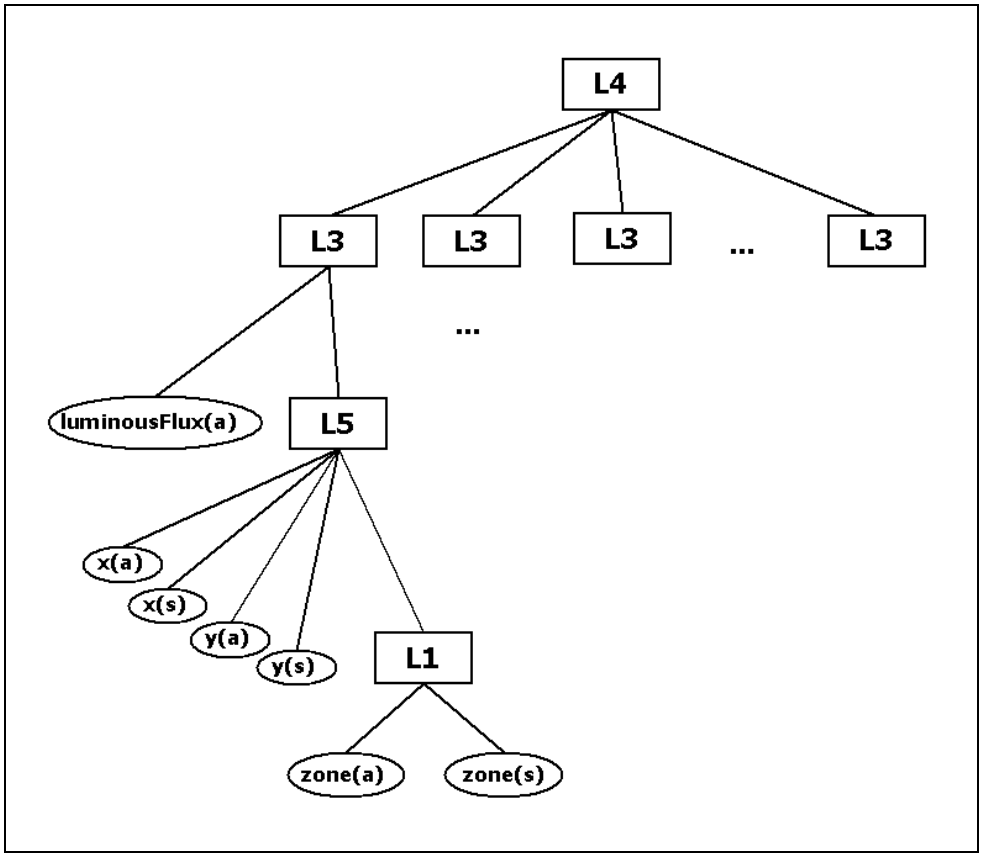

Figure 55. Call tree for the 2D Light Law Set

4.2.1.2.3. ***In a zone divided environment (least detailed)***

Going down further in the hierarchy of the detail level of the law sets defining the light effect we define the *OnOffLightLawSet*.

In some cases we can't estimate the luminous flux of a light source, or we don't know the exact position of the light source and/or the light sensor; in that case the Ambient Intelligent system fails to predict the theoretical value of the light sensor using the previous law sets, which are more detailed, that is why it tries to detect faults with the Boolean (or on/off) law set, which uses less information from the model, thus naturally giving less accurate fault detection results. The goal is no longer to match the theoretical value of what the light sensor is supposed to read, but to determine whether or not the status (activated/not activated) of the sensor corresponds to the status(on/off) of the light source that is in the same zone. So the fault detecting in that case is mainly finding out if the light sensor and the light source are activated or deactivated in the same time when at the same zone.

The mathematical functions of this law set are:

$$\text{sameZone}(s,a) = zone(s) \equiv zone(a) \tag{L1}$$

$$\text{booleanDirectLightExposure}(s,a) = \begin{cases} true & when\ \text{luminousFlux}(a) \neq 0\ AND\ \text{sameZone}(s,a)\ is\ true \\ false & when\ \text{luminousFlux}(a) \equiv 0\ OR\ \text{sameZone}(s,a)\ is\ false \end{cases} \tag{L6}$$

$$\text{ambientLightIntensity}(s) = \bigcup_{a \in LightEmitters} \text{booleanDirectLightExposure}(s,a) \tag{L7}$$

In **(L6)** we suppose that *on* status corresponds to the value *true*, and the *off* status corresponds to the value *false*.

(L7) To estimate the status of the light sensor when exposed to many light sources we apply a general OR Boolean function to all light sources, which means that it is enough to have on of the light sources on to expect the sensor to be activated (readings of the sensor are different from the value 0 or null).

The call tree of the Zone Light Law Set is as shown in **Figure 56**:

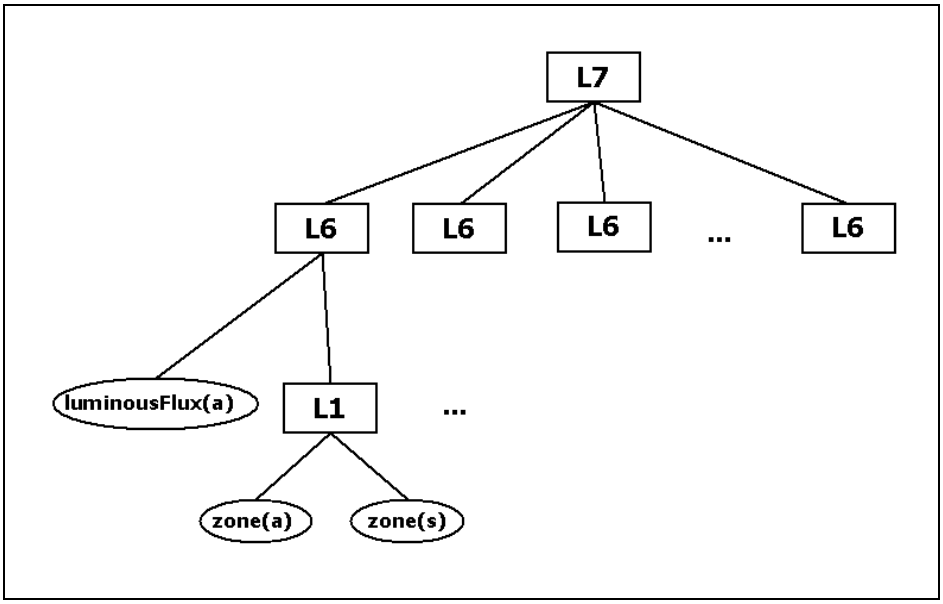

Figure 56. Call tree for the zone Light Law Set

4.2.1.3. **Light related entities in the concrete model**

From the description of light effect we can deduce the main entities to be introduced added to the concrete model. These entities are:

- *Light Effect*: instance of the abstract entity *Effect*.
- *Luminous flux*: instance of the abstract entity Effect Property.
- *Ambient Light Intensity*: instance of the abstract entity Measurable Property.
- *3D Light Law Set*, *2D Light Law Set* and *Zone Light Law Set*: instances of the abstract entity *Law Set*.
- *3D Position*, *2D Position* and *Zone*: instances of the abstract entity *Property*.

The relationships between these new entities are deduced from the abstract entities that hey instantiate. We can see these relationships in **Figure 57** below:

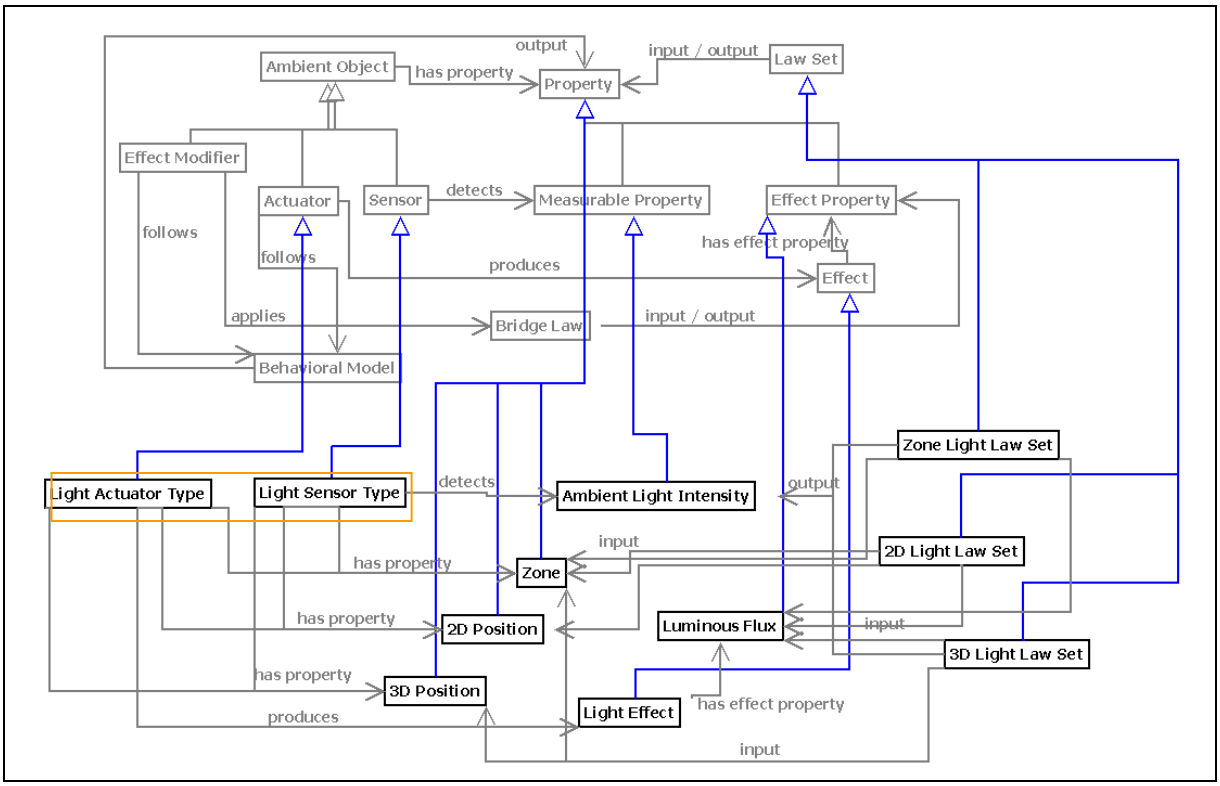

Figure 57. Entities of the concrete model after the definition of Light Effect

Note here that we added *Light Actuator Type* and *Light Sensor Type*, as instances of the abstract types, respectively, *Actuator* and *Sensor*. Note that we use generic names, such as *Light Actuator Type* and *Light Sensor Type*, at this level only for explanation purposes. We insist here that in order to have a better designed concrete model actual types of actuators and sensors should be instantiated, such as, for light actuator types, *Incandescent Light Bulb 40Watt*, *Fluorescent Light Bulb 80Watt*, etc. and, for light sensor types, *P-I-N Photo Diode*, *Infra Red Photo Transistor* etc.

According to these specific types the relationships with the other properties are drawn. In this case we have drawn all possible relationships for the *Light Actuator Type* and *Light Sensor Type*.

4.2.2. The Heat Effect

Having the preferred ambient temperature is also one of the main requests of ambient environments users. In our model we call this effect the "Heat Effect". Typically, the Heat Effect is produced by actuators of type Heaters or Air Conditioners (AC). It is characterized with, at least, the effect property heat emission, which can be of positive (heating) or negative (cooling) value. It is detected by the sensor type 'Thermometers'. The measured property by thermometers is the ambient temperature.

4.2.2.1. **Deduce temperature from heat emission – physical definition**

The incremental heat elevation is caused, according to enthalpy theory **[181]**, by total accumulated quantity of energy Q added to the system by an actuator generating that energy. The value of the energy Q is calculated using an integration of the instantaneous amount of power P that is generated by a resistor over time:

$$Q = \int_{t_i}^{t_f} P(t).dt \quad [joule] \qquad \textbf{(F1)}$$

In **(F1)**, t_i refers to the instant where the resistor started generating the power *P*, and t_f refers to the moment the resistor stopped generating the power *P*. From a modeling point of view these moments in time refer, respectively, to the start and end time of the Heat Effect having the property heat emission, whose value is *P*. To be able to perform discrete calculations, this integral is converted into a sum of instant power values over the same interval of time:

$$Q = \sum_{t=t_i}^{t_f} P(t) \quad [joule] \qquad \textbf{(F2)}$$

It is to be noted that with this method the current energy value is calculated using both, the current (at *t*) produced heat emission (power value generated by the resistor), and the previous (at *t-1*) calculated energy value. To calculate the ambient temperature of the air (or any other surrounding. e.g. water) from this energy formula we use enthalpy formulas **[182][183]**.

- The first formula that states that at a constant volume and pressure:

$$v.c = \frac{\Delta H}{\Delta T} \qquad \textbf{(F3)}$$

In **(F3)**

c is the milieu-specific heat capacity, which is the amount of heat required to change a chemical's temperature (air, water, oil, etc.). In physics this constant is called the volumetric heat capacity. For instance the volumetric heat capacity of air is 0.001297 $J.cm^{-3}.K^{-1}$. (See **Annex-B** for the complete list of volumetric heat capacities of different chemicals).

v is the total volume of the chemical to be heated (or cooled). For the case of air, this value (in cm^3) can be deduced via the dimensions (width, length and height) of the zone (e.g room) in which the calculations are performed.

ΔH is the enthalpy variation and ***ΔT*** is the temperature variation.

- The second formula states that, also under constant (atmospheric) pressure, the quantity of heat *Q* received by a system is equal to its enthalpy change ⊿*H*. So, from t_i to t_f, a body of volume *v* where the temperature (the value we want to evaluate by these calculations, which corresponds to the reading of thermometer) receives the amount of heat:

$$Q = \Delta H \qquad \textbf{(F4)}$$

So to calculate the value of the expected temperature reading of a thermometer at any moment in time, we need to estimate the temperature variation ⊿*T* between the start of the heating effect (t_i) and the moment of evaluation (t_f). We start by calculating the volume of the air exposed to the heating actuator (in the case of the air it is equal the dimensions of the room). Then we calculate the accumulated heating energy dispersed in the air using **(F2)**. Then using **(F3)**, while replacing ⊿*H* with *Q* **(F4)** value calculated in **(F2)**, we can calculate the temperature variation ⊿*T*.

4.2.2.2. **Deduce temperature from heat emission –mathematical modeling physical laws**

The functions described in the previous section are from a physical definition point of view; they are converted into laws and added to a Law Set. We call the resulting Law Set the *Ambient Temperature Law Set*. It is composed of the following laws:

$$\text{sameZone}(s,a) = zone(s) \equiv zone(a) \quad \textbf{(L1)}$$

$$\text{Energy}(s,a) = \begin{cases} heatEmission(a) \times elapsedTime & when\ \text{sameZone}(s,a)\ is\ true \\ 0 \quad when\ \text{sameZone}(s,a)\ is\ false \end{cases} \quad \textbf{(L8)}$$

$$\text{temperature}(s) = \left(\sum_{a \in HeatEmitters} Energy(s,a) \right) \div (v \times c) \quad \textbf{(L9)}$$

The value of *elapsed time* in **(L8)** is deduced by the Fault Detection and diagnosis system at run time.

In **(L9)** the value of *c* (the volumetric heat capacity) is a constant that can be deduced from the type of medium through which the effect is propagated. However we suppose, for now, that the value of *v* (total volume of the air) is a constant value provided by the Fault Detection and Diagnosis system.

In the call tree for the Ambient Temperature Law Set the *elapsedtime*, *v* and *c* are considered as leaves.

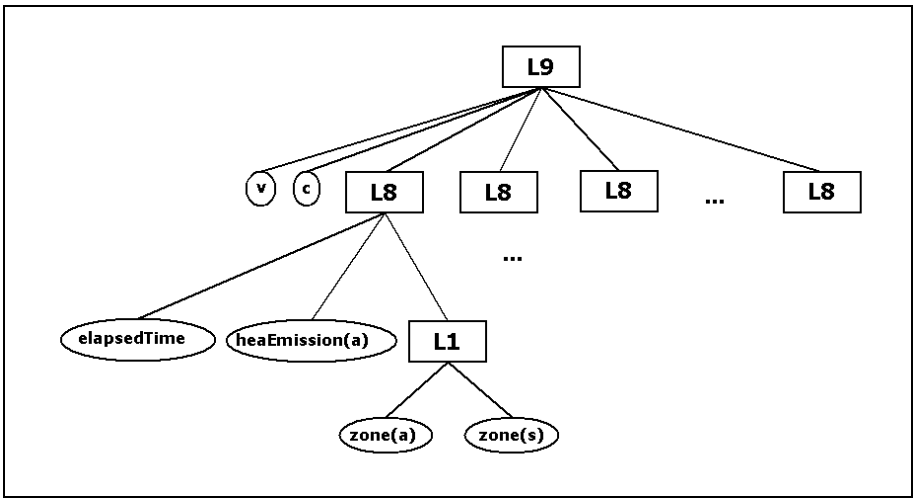

Figure 58. Call tree for the Ambient Temperature Law Set

4.2.2.3. **Deduce temperature from heat emission –the concrete model**

From the description of the Heat Effect we can instantiate a number of entities that we can add to the concrete model. These entities are:

Light Effect: instance of the abstract entity *Effect*.

Luminous flux: instance of the abstract entity Effect Property.

Ambient Light Intensity: instance of the abstract entity Measurable Property.

3D Light Law Set, *2D Light Law Set* and *Zone Light Law Set*: instances of the abstract entity *Law*.

3D Position, *2D Position* and *Zone*: instances of the abstract entity *Property*.

The relationships between these new entities are deduced from the abstract entities that they instantiate. We can see these relationships in **Figure 59** below:

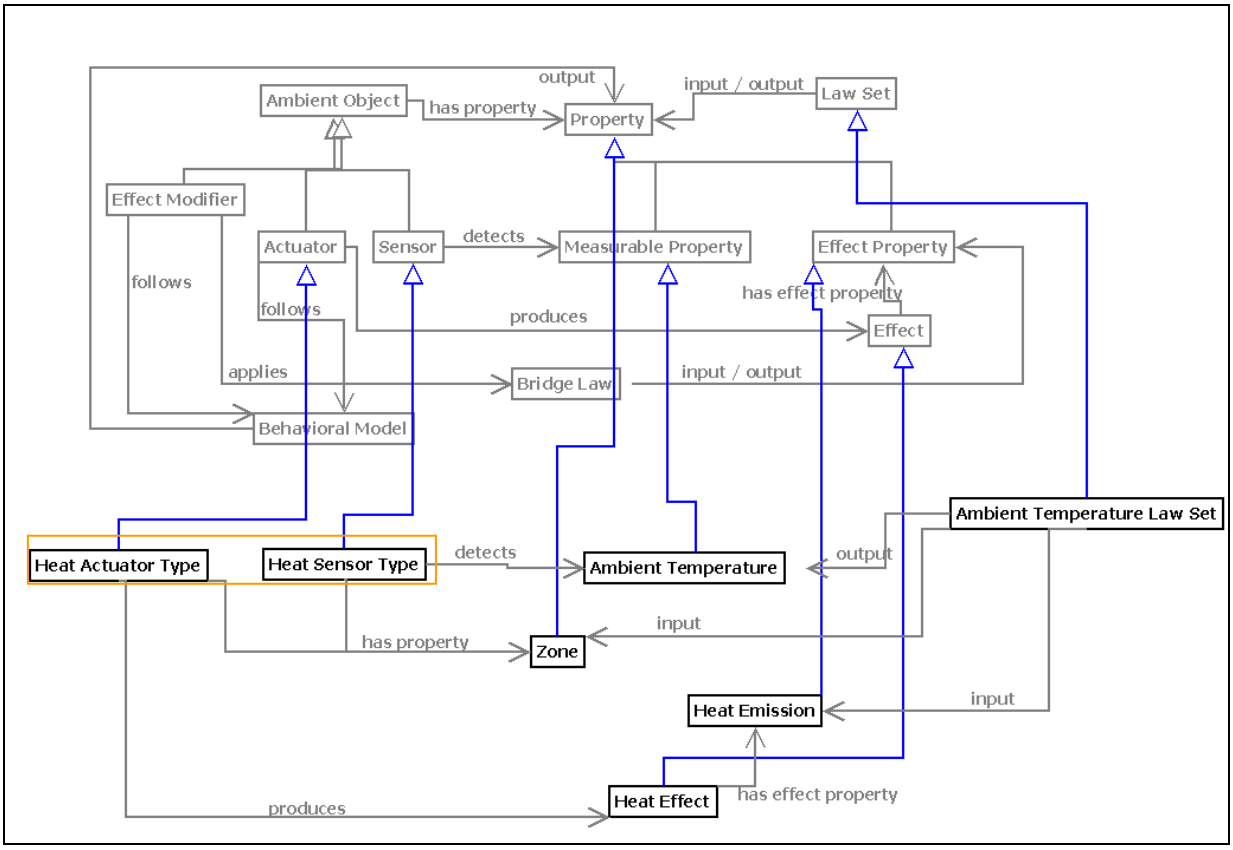

Figure 59. Entities of the concrete model after definition of Heat Effect

We note here that the same remarks about *Light Actuator Type* and *Light Sensor Type*, from the previous concrete sub model, are also valid for *Heat Actuator Type* and *Heat Sensor Type*.

As specific types of *Heat Actuator Type* we cite heating fan, air conditioner, electric fire place, etc. and the Heat Sensor Type is typically a thermometer.

Note that when the Heat Actuator Type is a type of device designed for heating the Heat Emission property value is positive, and when the Heating Actuator is for cooling the air the Heat Emission property value is negative.

4.2.3. The Water Flow Effect

Among the many automated services that an Ambient Environment provide to its users is to automatically prepare a bathtub. The command of preparing a bathtub usually triggers a physical phenomenon that can be modeled with the concept of Effect, which is the filling of the bathtub with water. Such task should be supervised to make sure it goes without errors (avoiding water leakage). In this section we study and model the Water Flow Effect. Note here that this effect can be applied to any "filling a container with a liquid" task that is done by the Ambient System, and that we only take the bathtub as an illustrative example.

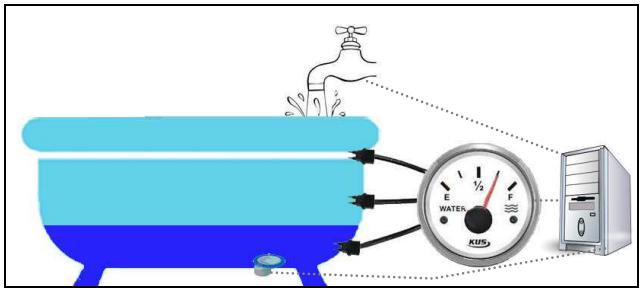

Figure 60. A simple Ambient Environment bathtub configuration

The main actuators for controlling the liquid level are flow actuator (the water tap), and the liquid drain (the bathtub drain). And we have one sensor to control the liquid level.

It is important to note here tha preparing a bathtub usually is associated to a preferable temperature of the water by the user. In the previous section we considered the heat effect in air; however it is important to note that when considering the heat effect in water (e.g swimming pool, bathtub) the dimensions of the zone containing the water do not necessary correspond to the volume of the water. That is another reason why we need a mathematical law allowing the evaluation of water volume.

Water flow effect makes the task of filling a container with liquid a diagnosable phenomenon, for instance when the system is filing a swimming pool with water at a certain level, and it is equipped with water level sensors, our fault detection and diagnosis framework needs to be able to detect any inconsistencies between the wanted level and the real level.

4.2.3.1. **Deduce liquid level from liquid discharge rate –mathematical modeling (the law sets)**

The idea is to define the *Liquid Flow Effect* so that it has the property *liquid discharge rate*. The latter is used with the elapsed time to deduce the expected amount of water that is supposed to be in the liquid container and compare that with the readings of the Liquid Level Sensor that is in the same container.

We call the mathematical set responsible for these calculations *Liquid Level Law Set*. The complete mathematical law set is as below:

$$\text{sameZone}(s,a) = zone(s) \equiv zone(a) \qquad \textbf{(L1)}$$

$$level(s,a) = \begin{cases} \text{dischargeRate}(a) \times elapsedTime & when\ \text{sameZone}(s,a)\ is\ true \\ 0 \quad when \quad \text{sameZone}(s,a)\ is\ false \end{cases} \qquad \textbf{(L12)}$$

$$totalLevel(s) = l_0 + \sum_{a \in WaterSources} level(s,a) \qquad \textbf{(L13)}$$

In **(L1)**, by same zone here we mean same water container (e.g swimming pool, bathtub, sink, etc.).

(L12) is associated with the water flow effect, which is produced by actuators of type water source and has the effect property discharge rate (liter per second) that is used in the

mathematical law. Multiplying the discharge rate with the elapsed time gives us the water level discharged by the current actuator.

(L13) considers the water quantity that is produced by the different actuators and sums up their produced water quantities. The result of **(L13)** is the total liquid level, which is comparable to the water level which is reading of the water level sensor. l_0 is the initial water level in the water container.

In the call tree of the Liquid Level Law Set (below), l_0 and *elapsedTime* are considered as leaves.

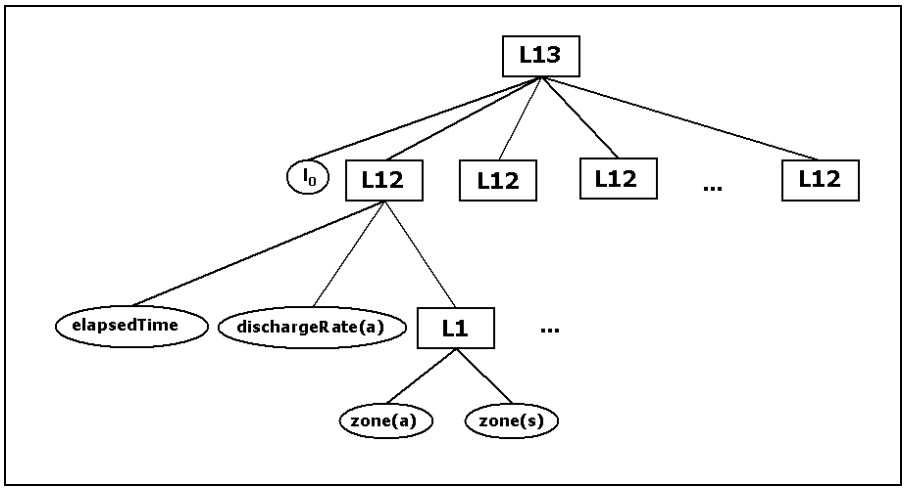

Figure 61. Call tree for the Liquid Level Law Set

4.2.3.2. **Deduce liquid level from liquid discharge rate –the concrete model**

From the description of the Liquid Flow Effect we can deduce a number of entities that we add here to the concrete model. These entities are:

- *Liquid Flow Effect*: instance of the abstract entity *Effect*.

- *Liquid Discharge Rate*: instance of the abstract entity Effect Property.

- *Liquid Level*: instance of the abstract entity Measurable Property.

- *Liquid Level Law Set*: instance of the abstract entity *Law*.

- *Zone*: instance of the abstract entity *Property*.

The relationships between these new entities are duplicated based on the abstract entities that they instantiate. The new concrete model is shown **Figure 62**:

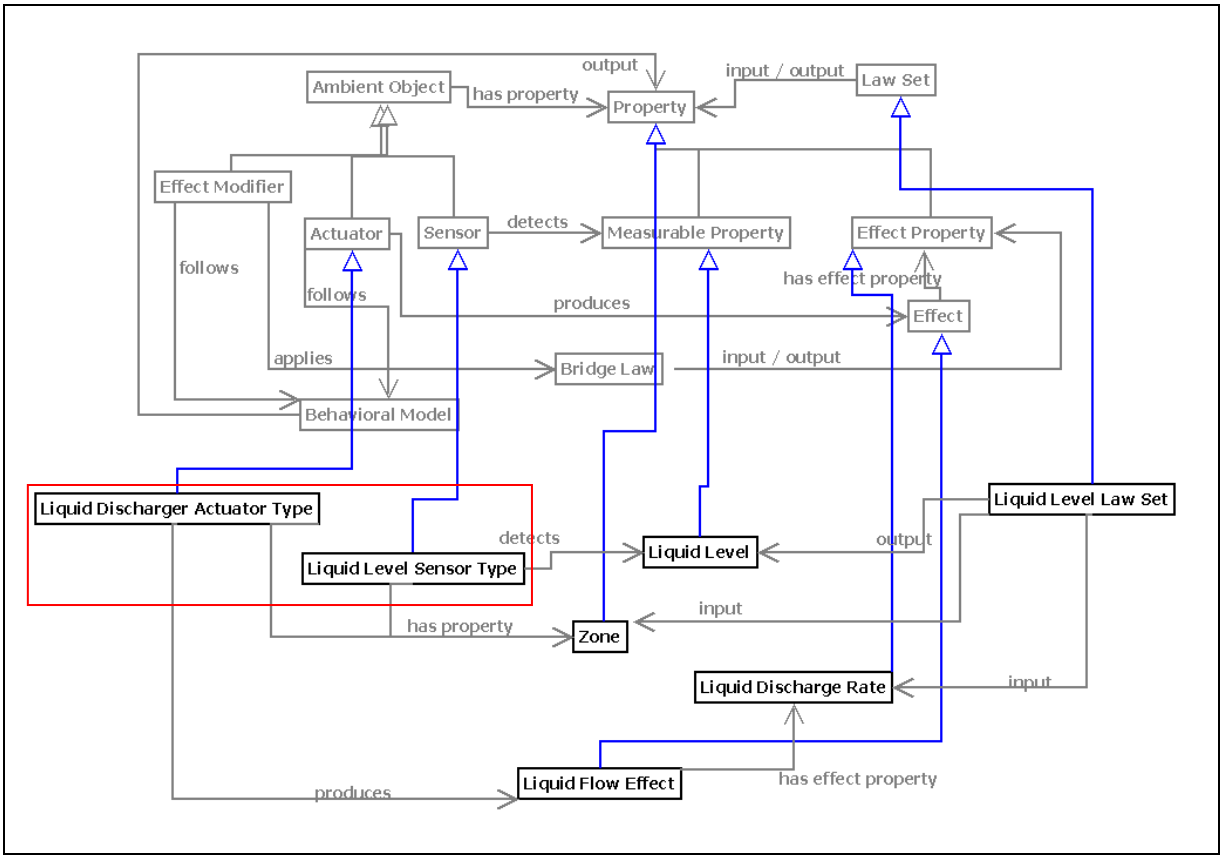

Figure 62. Entities of the concrete model after definition of Liquid Flow Effect

At instance level there are mainly two kinds of Liquid Discharge Actuator types: Those whose *Liquid Discharge Rate* are positive such as water taps or water sources, and those whose *Liquid Discharge Rate* are negative like water drains.

4.3. The Concept of Sensor (Effect Receiver)

The Sensor is the component that allows the Ambient Intelligent System to be aware of what is happening and of the state of the Ambient Intelligent Environment. Our fault detection approach is based on using the available sensors in the Ambient Intelligent Environment to monitor the proper functioning of the different controllable devices (actuators, modifiers) and to determine, when a fault is detected, the device(s) or external factors causing the fault.

4.3.1. Meta-model of the concept of Sensor

From the Abstract Model's (more precisely from the Effect's) point of view, the entity "Sensor" is the Effect receiver. In fact a sensor has a "receive" relationship with the entity "Measurable Property" as shown in

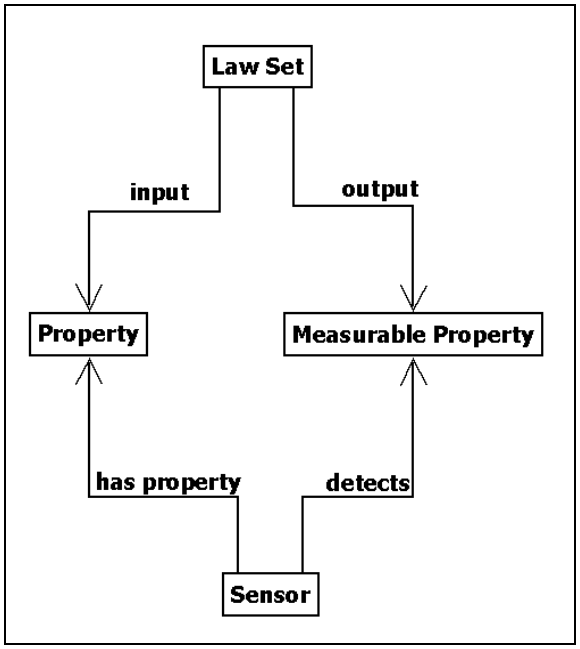

Figure 63. Entities from the abstract model connected to "Sensor"

4.3.2. The Measured value (*Measurable Property*)

The entity *Measurable Property* is the key to the Fault detection task. The Measurable Property entity represents the supposed reading of the Sensor that "detects" it.

At instance level the entity of type *Measurable Property* contains the value of the reading of the sensor that it is connected to (with the relation "detects").

Once the Prediction Model is created, it is able, via the laws composing the Law Set (note the latter's "output" relation with *Measurable Property* in **Figure 63** and in the Concrete Models in **Figure 57**, **Figure 59**, and **Figure 62**), to assign a theoretical second value for this entity.

It is based on the comparison between the predicted theoretical value and the actual value for the same entity *Measurable Property*, that the Fault Detection task is done.

4.3.3. Sensor Properties

Like other entities of type "Ambient Object", at the Concrete Model level, sensors can have other properties. Next we list the most useful (in some cases necessary) for a better Fault Detection and Diagnosis.

- Localization properties (also used for other Ambient Object instances)

The localization properties are also used for other Ambient Objects in order to localize them in the Ambient Environment Space. They are important as they are crucial to evaluate some important physical quantities such as "distance". The localization property is particularly important for sensors because when they are missing, unlike for actuators in which case the latter are simply ignored in the Prediction Model, the quality of the fault Detection and Diagnosis results are significantly lowered. To better see this, considering a sensor that is exposed to the effects of 3 actuators, if the localization properties are well defined for all entities except for 1 actuator, the sensor performs the fault detection task ignoring that actuator. However if all the entities have well defined localization properties except for the sensor, physical laws cannot be applied for all entities.

Among these properties we cite *Zone*, *2DPosition*, and *3DPosition*. When no localization property is given to all Ambient Objects, the framework assumes that they are all in the same "Zone" and the appropriate Law-Sets (usually the least detailed) are selected for generating the Prediction Model. For an Ambient Environment that is composed of multiple zones, we recommend having at least the localization property *Zone* (in reality it is up to the designer to chose the name of the property Zone, it can have different name such as *Room*, *Area*, *Range*, or simply *Zone*).

- Tolerance

The tolerance value is deduced from the accuracy property that is usually given by the sensor's constructor (see accuracy definition in **3.2.3**). It is very useful to consider the tolerance value when comparing the theoretical value versus the actual reading of a sensor, in order to avoid false fault-detection alarms.

4.4. The Concept of Actuator (Effect Producers)

In an Ambient Intelligent Environment, an actuator is the component that physically acts upon the environment. In our framework we represent these actions in the form of effects. Naturally the actuator is the producer of these effects.

4.4.1. Meta-model of the concept of Actuator

In **Figure 64** we see how these entities (actuator and effect) are connected.

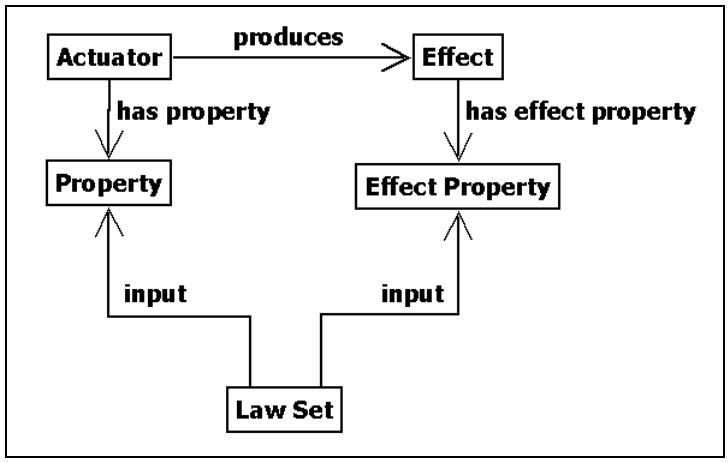

Figure 64. Entities from the abstract model connected to "Actuator"

An actuator produces one or more effects; each of these effects has one or more effect properties. These properties represent a quantification of the physical property of the effect that can be observed in the environment. At instance level, these effect properties and actuator properties (such as x y position, tolerance value, etc.) are used as inputs in the laws constituting the law-set that mathematically models the effect. The values of these properties can either be static, in which case they keep their defined values indefinitely (the tolerance value for instance), or they can be dynamic, in which case their value changes according to the current state of the actuator. The latter case can be modeled via behavioral model.

4.4.2. The Actuator's Behavioral Model

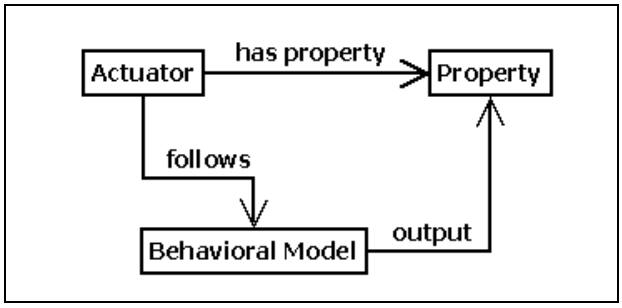

Figure 65. The actuator's behavioral model

In **Figure 65** is a zoom-in on the entities from the abstract model that defines the relationship between the entity *Actuator* and the entity *Behavioral Model*. Entities of types *Actuator* follow a *Behavioral Model*. In fact, at instance level, these behavioral models define and update the values of entities of type *Property* that are associated to the actuator. Behavioral Models are also used to define the values of entities of type *Effect Property* that are associated to the *Effect* generated by the instances of *Actuator*.

In practice we use two types of finite state machines to describe components behaviors: classic finite state machines and timed finite state machines.

It is important to note that even though we chose finite state machines as behavioral models here; our approach does not impose a specific formalism to describe object behavior, so the designer is free to use different types of behavioral models, for instance Petri Nets.

4.4.2.1. Classic Finite State Machines

As classic finite state machine we use a subset of UML state machine. The labels on the transitions have the format "*event*/*action*". Where *event* is what triggered the transition to the next state, and *action* is what is to be executed with the transition. We use these Finite State Machines to model devices behavior. A typical Finite state machine for a device that has only on and off states is the following:

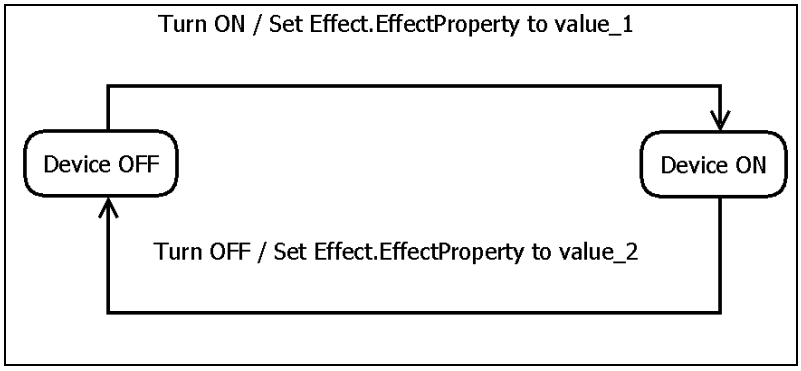

Figure 66. On Off only device's FSM

The action that is done in the transition is to change the value of an *Effect Property* related to the device. There are two transitions in this finite state machine:

- From the state *off* to the state *on*, during which the Effect produced by the device changes value to value_1.

- From the state *on* to the state *off*, during which the Effect property of the effect produced by the device changes value to value_2.

This is the default finite state machine for most simple on/off controlled devices such as incandescent light bulbs.

Let's suppose we have a 40 Watts incandescent light bulb. Incandescent light bulbs have luminous efficacy of 15 lumens per Watt (see **Table 3**), which means the luminous flux would be:

40 W x 15 lm/W = 600 lm

The Finite state machine that models the behavior of such a light bulb would be as described in **Figure 67**:

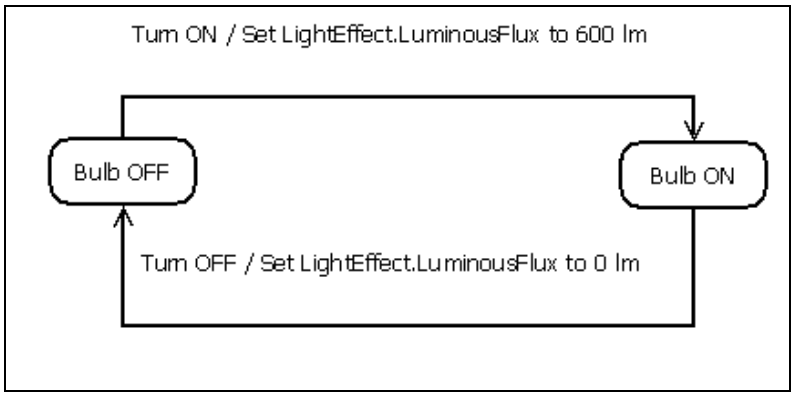

Figure 67. Finite state machine for a 40 Watts incandescent light bulb

4.4.2.2. **Timed Finite State Machines**

In fact, effects model physical phenomena, and frequently the latter depends on time variables. In fact when examining the behavior of the actuators in an Ambient Intelligent Environment, we notice that, many times, from the time actuators are activated, the physical impact takes a certain delay before it can be observed. The duration of these delays vary depending on:

- The nature of the physical phenomena: for instance after turning on a heater, the heat effect that is supposed to be produced is not detectable until a certain time has passed; the length of this time is defined by heat transfer laws.

- The type of the actuator: for example compact fluorescent lamps (CFL) take a certain time before they emit their maximum amount of light. This warming up time is usually defined by the constructor.

To take such properties into account we use Time Finite State Machine, which are a variation of Finite State Machine that have integer boundaries for time guards and performs a time reset operation at every transition **[184]**.

A typical example of actuators whose behavior is described using timed finite state machines are the light actuators of type Compact Fluorescent Light (CFL) Bulbs. In fact, when we turn a CFL light bulb on, it needs a certain time until it reaches its maximum luminosity. This is called the heating time. The heating time is different between different types of CFL light bulbs.

In **Figure 68** we see a timed state machine that better describes the behavior of a CFL light bulb. T represents the elapsed time since the beginning of the current transition.

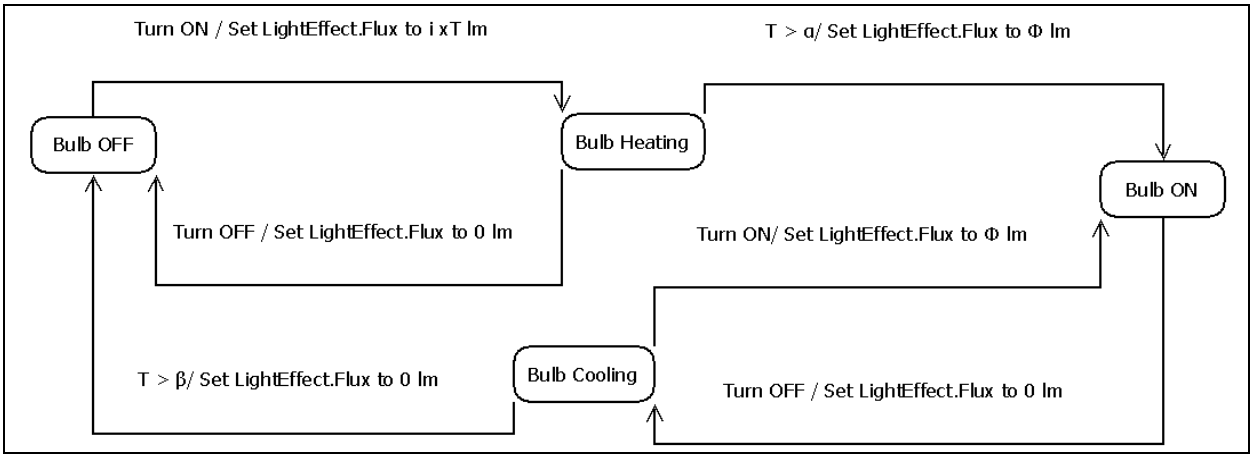

Figure 68. Compact Fluorescent Light Bulbs Finite State Machine

For every instance of CFL light bulb we will have different values for:

Φ: the value of the luminous flux calculated using **Table 3**.

α: the heating up time; the time necessary for the CFL bulb to reach its maximum luminous flux value.

β: the cooling down value; the time necessary for the CFL bulb to cool down. It is important to calculate this time because if the CFL bulb is reactivated before the cooling time has passed it will not behave as it normally would when turned on (go through Bulb Heating state). Instead it will transition directly to the Bulb On state.

i: the rate of light flux augmentation every unit of time during the warming up phase.

These entities are defined as properties for the *Actuator* type *CFL Bulb* in the *concrete model*.

The plot in **Figure 58** illustrates a scenario where we turn on a cooled CFL light bulb. Then we turn it off, and before it cools down again we turn it on.

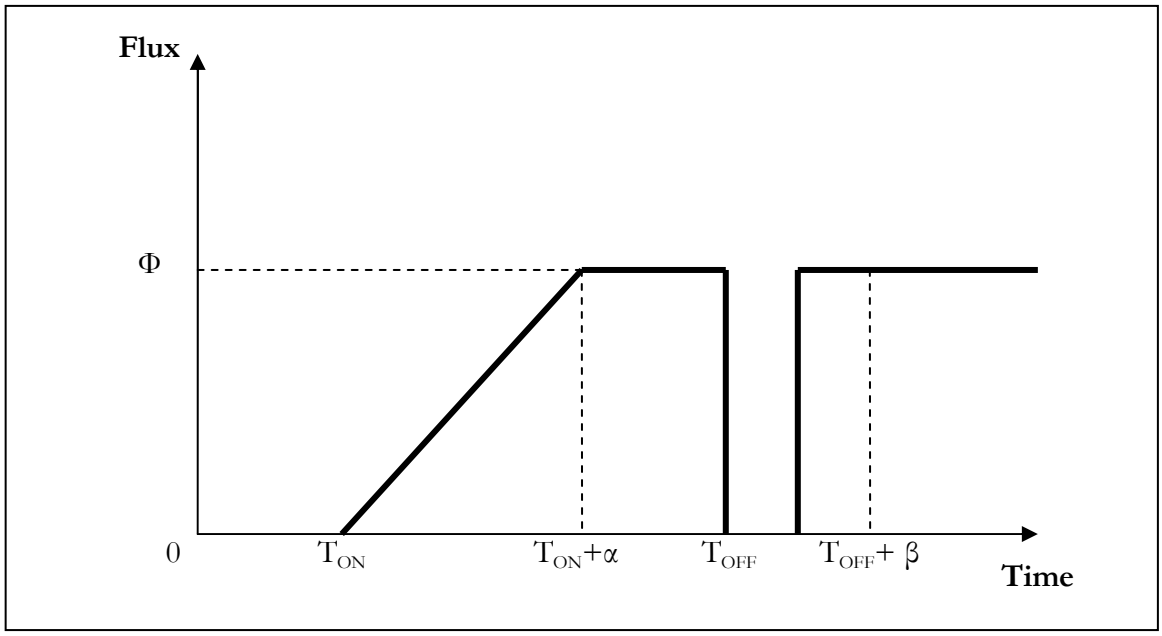

Figure 69. Light flux value according to time for a typical CFL light bulb

We can observe the gradual augmentation of the value of the light flux at the heating phase (from the turning on moment T_{ON} until $T_{ON}+\alpha$). After turning off the CFL bulb (at T_{OFF}), we

turn it on again before the cooling time elapsed ($T_{OFF}+\beta$), we notice then the abrupt augmentation in light flux.

We note here that in some cases more advanced temporal behaviors ought to be modeled. In fact our FDD framework also allows the use of a variation of temporal logic that has real time constraints, which is Metric Temporal Logic **[185]**. The latter allows the definition of lower and upper bounds and ranges for relative and real time constraints. With MTL, we can model deadlines between environment events and corresponding system responses when describing the behavior of real-time system.

Considering the following scenario as an example **[186]**: {when an "alarm" is triggered, we must "shut down" the system after 10 time units unless an "all clear" signal is received before that}; such constraint is represented as follows:

$$\Box(alarm\ (\Diamond_{(0,10)}\ allclear \vee \Diamond_{\{10\}}\ shutdown))$$

where $\Diamond_{(0,10)}$ means sometime in the next 10 time units

and $\Diamond_{\{10\}}$ means in exactly 10 time units

4.5. The Concept of Effect Modifier (an Effect Receiver and Producer)

The *Effect Modifier* is a special case of Ambient Objects. An *Effect Modifier* is not an actuator in the sense that it does not generate a physical phenomenon itself. And it is not a sensor in the sense that it does not have sensing capabilities to know the actual value of the physical phenomenon that it is exposed to. However a modifier is a device that is part of the Ambient Intelligent Environment and that is controlled by the Ambient Intelligent System. This device is usually an object that connects two or more zones of the Ambient Intelligent Environment, such as a door, a window, window blinds, etc., and that relays a physical phenomenon between the zones it connects.

The most important characteristic (in the sense that it affects the result of a fault detection and diagnosis results) of an *Effect Modifier* is that it modifies a physical phenomenon that is already produced by an actuator in the environment when the physical phenomenon is relayed to a neighboring zone.

4.5.1. The Meta-model of the concept of Effect Modifier

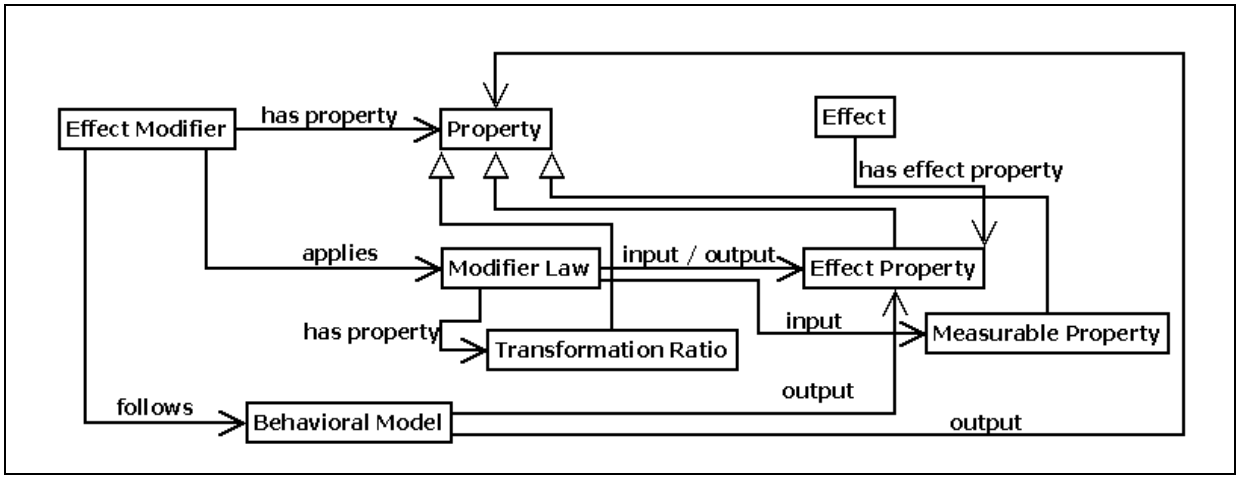

Figure 70. Entities from the abstract model connected to "Effect Modifier"

The entity of type Modifier has a transformation law associated (*Modifier Law*) that defines the change that it makes on the physical phenomenon it relays from one Zone to another. The Modifier Law has (in addition to the Measurable Property it is exposed to and the Effect Property it would relay), one or more, property called the *Transformation Ratio*. For instance when defining a *Light Effect Modifier* in the *Concrete Model* the *Modifier Law* would be:

$$\mathbf{LuminousFlux} : \Phi = \gamma \,.\, I$$

where:

γ: is the transformation ratio, which is a *Property* of the *Effect Modifier*. The value of the transformation ratio changes according to the current state of the *Effect Modifier*, which is defined by the behavioral model; see **4.5.2**.

I: An input to the Modifier Law in this case, it is the amount of *Light Intensity* that the *Effect Modifier* is exposed to. This value is calculated using the *Law Set* that is used to predict the reading of a *Light Sensor* in a *Zone*.

Φ: An output of the Modifier Law, it is the value of the *Luminous Flux* (*Effect Property*) of the *Light Effect* that the *Effect Modifier* is relaying to a neighboring *Zone* to that in which the value of **I** was estimated.

The connection between this Modifier Law and the Light Effect is made by matching the type of the Properties that are input and output of the Modifier Law. A modifier can relay different types of Effect. A modifier can then have different Modifier Laws to define the different effects it relays.

When we consider an Effect Modifier between Zone1 and Zone2, if we want to estimate the amount of an Effect that is relayed from Zone1 to Zone2, the Effect Modifier would be considered as a Sensor in Zone1, in order to apply the Law Set that estimate the exact amount of the Effect it is exposed to at its position. Once the value is calculated (for instance the light intensity **I**) to which the modifier is exposed, we can apply the Modifier Law to calculate the value of the Effect (for instance the Luminous Flux Φ) it produces. The Effect Modifier is now considered as an Actuator in Zone2 that produces the corresponding Effect. For instance a window is considered by a light sensor as light source even though it does not generate the light (it produces into the Ambient Environment) itself.

It is important to note here that when estimating the value of a physical property (for instance the light intensity **I**) to which a certain modifier is exposed, we do not consider the contribution of the same Effect Modifier's produced Effect (for instance Luminous Flux) when evaluating the laws to calculate the received physical l property. In fact doing so can result in an infinite calculation loop when a modifier considers its own contribution to the final calculation results.

A consequence of that would be a small difference when estimating the values of **L4**, **L7**, **L9**, **L13** by a real Sensor and by an Effect Modifier that considered as a Sensor. The difference is that when applying such laws (***L4*** as an example) for a **Sensor** we consider that *LightEmitters* in $\sum_{a \in LightEmitters}$ refer to all devices that produce the Light Effect (i.e. **Actuators and Effect Modifiers**), **however** when applying the same law for an **Effect Modifier** considered as a Sensor (to estimate the value of a physical property) the *LightActuators* refers to **Actuators and Effect Modifiers <u>except</u> for the Effect Modifier that is doing the calculations**.

- **Special case**:

In some cases we can have a Sensor that is in the same position (or close enough) to an Effect Modifier. In that case the latter's reading can be used to determine the value of a measurable physical property (like the Light Intensity) it is receiving from the Zone where the Sensor is located. A typical example of this is an outside Sensor that reads an external physical property (such as light intensity or temperature). Such sensor can be used for instance to estimate the amount of light is coming through a Window type Modifier, without having to use the law set to estimate the light intensity coming from the outside at the window (which is impossible in fact since the outside effect sources are not known neither controlled by the system). In order to implement such solution, the corresponding entities in the Concrete Model should inherit the following extra relation between the Effect Modifier and the Sensor:

`EffectModifier-trustedSensor-Sensor`.

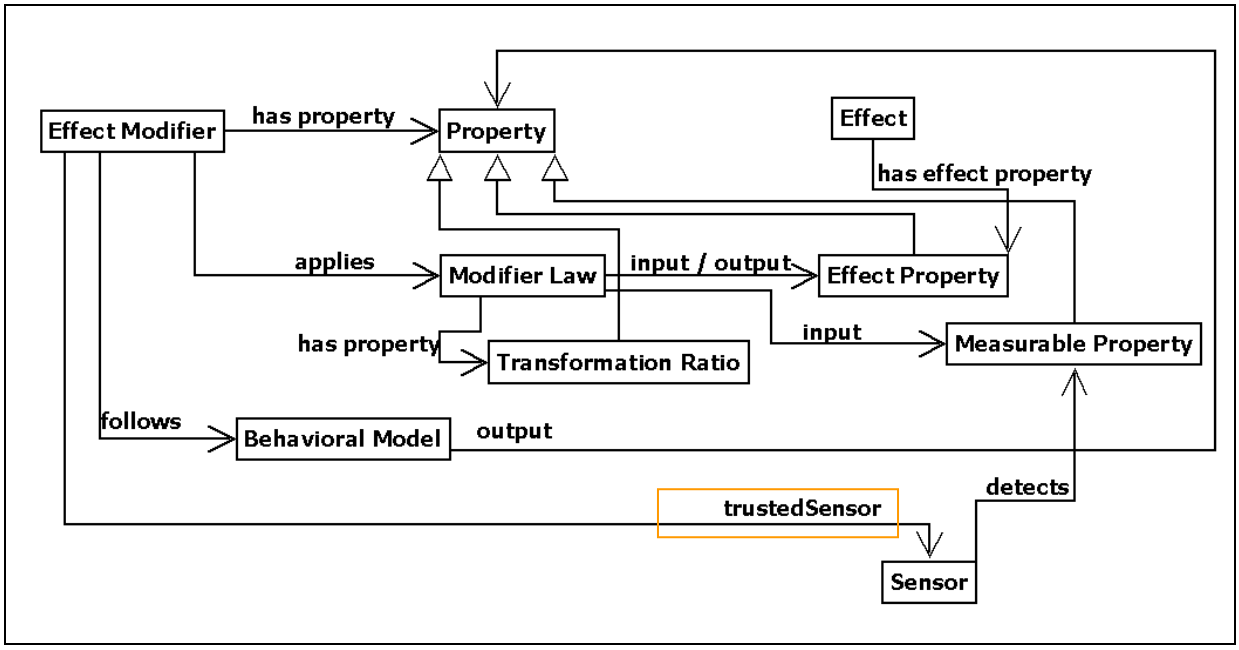

Figure 71. Entities from the abstract model connected to "Effect Modifier" (special case when a Sensor is linked to a Modifier)

When such relation (as shown in **Figure 71**) is instantiated between an instance of Effect Modifier and an instance of Sensor, the value of the Effect Property (for instance I for Light Effect) that the Effect Modifier is exposed to (the input of the Modifier Law) is not calculated via the appropriate Effect's Law Set, but instead it is deduced directly from the Sensor's readings. Such a solution leads to less calculations and it is particularly useful when the Effect Modifier is relaying an Effect from a non controlled Zone where the devices and components are not controlled by the system (for instance: the outside). It is important to note here that it is up to the system designer to choose such a solution for special cases.

Even though this solution manually connects Effect Modifiers to Sensors, Actuators are not considered, and are not concerned with this solution, thus the decoupling between Actuators and Sensor is still verified. Moreover, one would argue that Effect Modifiers (e.g windows, doors, glass walls, etc.) are generally not dynamic components, in the sense that they are not usually discovered, removed, or moved at run-rime.

4.5.2. Effect Modifier's Behavioral Model

The Effect Modifier's Behavioral Model governs the changes that happen to the values of the Modifier's Properties according to the current state of the Effect Modifier. In particular the behavioral model defines the value of the *Transformation Ratio* value used in the *Modifier Law*. The behavioral model can also control the value of other Effect Modifier's Properties. That is why, when designing the Concrete Model, we keep the entity *Behavioral Model* linked to the abstract entity *Property* (super class of *Transformation Ratio*).

In the same way that an Effect Modifier would have different Modifier Laws for the different Effects it relays, the Effect Modifier would have different Behavioral Models that define the modification ratio value for each Modifier Law.

A typical behavioral model for a two-state door (open-closed) that relays the totality of the Light Effect it receives from one Zone to the other is depicted on **Figure 72** finite state machine:

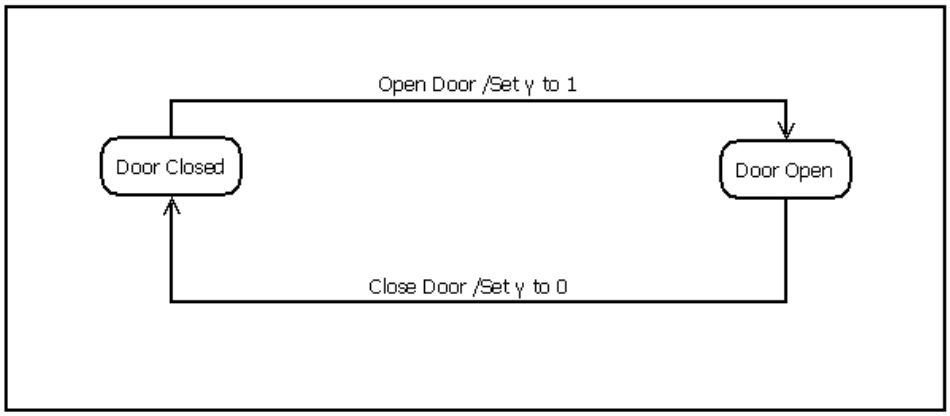

Figure 72. A two-state door finite state machine

So when the door is opened the transformation ratio for the Light Effect is equal to **1**, which means that the door would relay the totality of the Light Effect it receives from a Zone to the neighboring Zone. And when the door is closed it would not relay the Light Effect at all, which would be modeled (to respect the Modifier Law definition) by a Light Effect that has the Light Flux value of **0**.

4.6. Algorithm for building the prediction model from the conceptual models

In this section we explain the models definition syntax and we present our fault detection and diagnosis algorithm in an informal high-level pseudo-code.

4.6.1. The Prediction Model

Once we have our Concrete Model and the Instances well defined we can create our Prediction Model. The algorithm that generates the Prediction Model and the syntax in which it is defined in the framework are described in **Chapter 5**: Implementation. The prediction Model contains:

- The instances for the actual components of the Ambient System with their properties' values in a given configuration.

- Behavioral models for active components (Actuators and Effect Modifiers) allowing the update of their state, and by consequence the update of the latter mentioned properties' values.

- The Mathematical Model composed of sensors' call trees generated from law sets. In fact, the prediction Model applies the laws in the Law Sets on the actual instances, so the laws that have iterative expressions, such as $\sum$ in **L4**, are expanded according to the existing instances. For example if we have an Ambient Environment described in a two dimensional space, composed of one zone and equipped with one light sensor (sensor_1) and two light bulbs (bulb_1 and bulb_2). The Law Set for the Light Effect in a two dimensional space is the following:

$$\text{distance}(s,a) = \sqrt{(x(s)-x(a))^2 + (y(s)-y(a))^2} \quad \textbf{(L5)}$$

$$\text{directLightExposure}(s,a) = \frac{\text{luminousFlux}(a)}{\text{distance}(s,a)^2} \quad \textbf{(L3)}$$

$$\text{ambientLightIntensity}(s) = \sum_a \text{directLightExposure}(s,a) \quad \textbf{(L4)}$$

Note that we eliminated L1 (the zone verification Law) from the Law Set in this example because we suppose that the Ambient Environment is composed on one single zone.

Here's a list of the properties (on an instance level) that are going to be used in the evaluation of this Law Set:

sensor_1.x: the x coordinates of sensor_1

sensor_1.y: the y coordinates of sensor_1

sensor_1.i: the reading of the light sensor indicating the value of the Ambient Light Intensity around it

bulb_1.x: the x coordinates of bulb_1

bulb_1.y: the y coordinates of bulb_1

bulb_1.flux: the value that bulb_1 produces of its Light Effect's property Luminous Flux

bulb_2.x: the x coordinates of bulb_2

bulb_2.y: the y coordinates of bulb_2

bulb_2.flux: the value that bulb_2 produces of its Light Effect's property Luminous Flux

The call tree, in the Prediction Model, for this Law Set applied to sensor_1, and using the instances we have defined would look as follows:

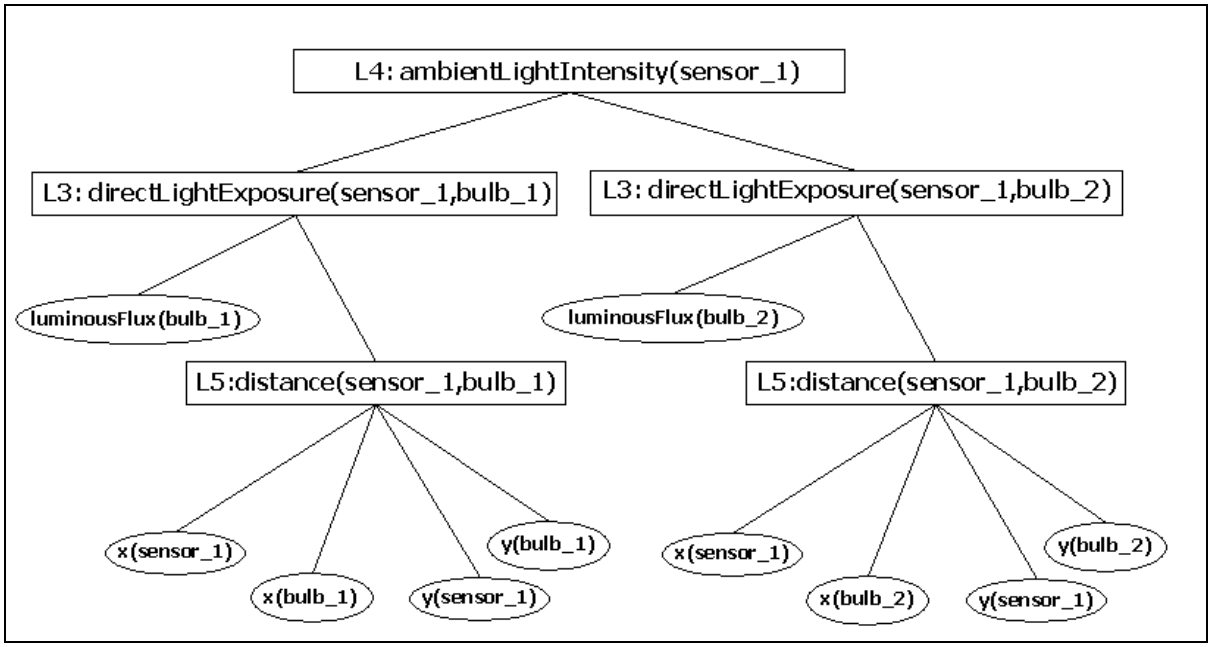

Figure 73. Call tree of 2D Light Law Set in the Prediction Model

In the previous call tree we have:

```
x(sensor_1)= sensor_1.x
y(sensor_1)= sensor_1.y
x(actuator_1)= actuator_1.x
y(actuator_1)= actuator_1.y
distance(sensor_1,bulb_1)=
sqrt([x(sensor_1)-x(bulb_1)]^2+[y(sensor_1)-y(bulb_1)]^2)
luminousFlux(bulb_1)= bulb_1.flux
directLightExposure(sensor_1,bulb_1)=
luminousFlux(bulb_1)/distance(sensor_1,bulb_1)^2
x(actuator_2)= actuator_2.x
y(actuator_2)= actuator_2.y
distance(sensor_1,bulb_2)=
sqrt([x(sensor_1)-x(bulb_2)]^2+[y(sensor_1)-y(bulb_2)]^2)
luminousFlux(bulb_2)= bulb_2.flux
directLightExposure(sensor_1,bulb_2)=
luminousFlux(bulb_2)/distance(sensor_1,bulb_2)^2
ambientLightIntensity(sensor_1)=
directLightExposure(sensor_1,bulb_1)+directLightExposure(sensor_1,bulb_2)
```

where sqrt is the square root function and x^y is x to the power y.

Note how **L4** (the summation operator) is expanded to add the contribution of each light bulb (bulb_1 and bulb_2) in Luminous Flux to the final Ambient Light Intensity around the sensor (sensor_1).

When evaluating the previous call tree, the Prediction Model will replace all the instances' properties by their respective values. These values can be defined as fixed property values (e.g tolerance, position of non movable objects, etc.), or dynamic property values (eg. light flux, transformation ratio), in which case they would appear on the behavioral models of their objects' instances.

The Prediction Model is updated when a new object is added (new instance) to the Ambient Environment or removed from it. However we do not need to update the whole Prediction Model when some properties' values change, as the instances and their relationships composing the model did not change, thus we can use the same model (and call trees) with different values to perform Fault Detection calculations.

Technically the Prediction Model is a local model created and contained (and constantly updated) in the Prediction Engine (see **Figure 51**). The constant data flow that updates the properties' values in the Prediction Model is governed by the behavioral models (in our case the finite state machines) of the corresponding objects. So not only the Prediction Model is a **heterogeneous** model, it is also a **dynamic** model that is executed at run-time in order to perform the Fault Detection task. The fault detection algorithm is detailed more in **Chapter 5**: Implementation.

4.7. Conclusion

In this chapter we have detailed "AmILoop", our Fault Detection and Diagnosis framework. We depicted its general architecture. Our framework is composed of different types of models that describe different aspects of the framework. In fact in the framework we described the structure of the components (their relations and their hierarchy) as well as their behaviors. We also introduced the different mathematical models that define some physical "effects". We also described how all these models are used by the prediction engine to generate a Prediction Model allowing the calculation of what sensors are supposed to read. In the next chapter we propose an implementation of our Fault Detection and Diagnosis Framework. We also detail the way our Fault Detection approach was integrated into a real Ambient Intelligent System. Then, in order to test more scenarios, we explain how we use the implemented framework with the heterogeneous model simulator ModHel'X in order to perform real-time model simulations.

Chapter 5:

Implementation

Chapter 5.

Implementation

In this chapter we present the implementation of our Fault Detection and Diagnosis (FDD) framework. At its heart lies a program that analyses the changes that occurs in the environment. Each time a significant change happens, it computes a new *prediction model* suited to the new configuration of the relevant entities (sensors, actuators, and their respective locations inside a building). The prediction model allows one to determine the values that should be reported by the sensors if everything worked properly. One can then easily compare these predicted values with the actual values to perform fault detection.

The prediction model is of heterogeneous nature: it contains both mathematical computations and finite state machines for representing the state of objects. Therefore we use a heterogeneous modeling framework to describe and execute this model. Among the heterogeneous modeling tools introduced in Section **3.4** we chose to use ModHel'X, a tool that is being developed at Supélec. This tool executes the prediction model at runtime, thereby providing the predicted values for sensor readings.

The implementation was developed within a European project called CBDP (Context Based Digital Personality) **[187]**. The project investigated several scenarios related to ambient intelligence, especially Ambient Assisted Living (AAL). We conducted tests using a simulated environment, but the same implementation could be used as is for real-scale experiments.

This chapter is organized as follows. Section 5.1 introduces the context for the implementation, the CBDP project. Section 5.2 is an introduction to ModHel'X, the underlying model execution engine that we use. Section 5.3 describes the software architecture of our FDD framework AmILoop, and it explains how the prediction model is built. Section 5.4 deals with the execution of the prediction model. It is mainly a short description of our simulation environment.

5.1. The Context Based Digital Personality project

CBDP (Context Based Digital Personality) was a European CELTIC project that ran from April 2009 to April 2012. It involved several French, Spanish and Turkish partners around the design and implementation of a framework for the creation of ambient-intelligent applications. It introduces the concept of Digital Personality to represent the preferences of users. We were mainly involved with the applications to Ambient Assisted Living (AAL), but other application domains were considered, such as digital TV guides or assistants for construction workers.

The CBDP framework is built around an ontology, and it uses software components written in Java and deployed over OSGi (Open Services Gateway initiative framework) **[188]**.

5.1.1. CBDP's AAL ontology

The approach of CBDP is based on ontologies to capture domain knowledge, and to perform run-time tasks such as the interconnection of devices, the exchange of data, the execution of system tasks, and fault detection and diagnosis. CBDP's AAL ontology is defined using the OWL language **[189]**, which is based on the Resource Description Framework (RDF) **[190]**. The knowledge in RDF is represented in the form of triples (also called statements) of the form of **`subject predicate object`**.

As depicted in **Figure 74**, the main domains of the CBDP ontology are:

- *Device*: the entities of this domain are based on the DogOnt ontology **[191]** that has been simplified, while keeping the modeling axes of typology, functionality and state.
- *Digital Personality*: composed of the entities: *Person* representing a human user, and *Digital Personality* that stores the user's preferences in order to personalize the services offered to him/her.
- *Location*: this entity is important since most of the services offered by the CBDP framework (in particular the AAL applications) must know the position of the user (for instance inside/outside the house, in the bedroom/in the kitchen, etc.) and of the devices (sensors and actuators).
- *Time*: entities of this domain are imported from W3C's existing Time Ontology **[192]**.
- *Fault Detection and Diagnosis*: the entities of the diagnosis framework described in Chapter Chapter 4 can be inserted in CBDP.

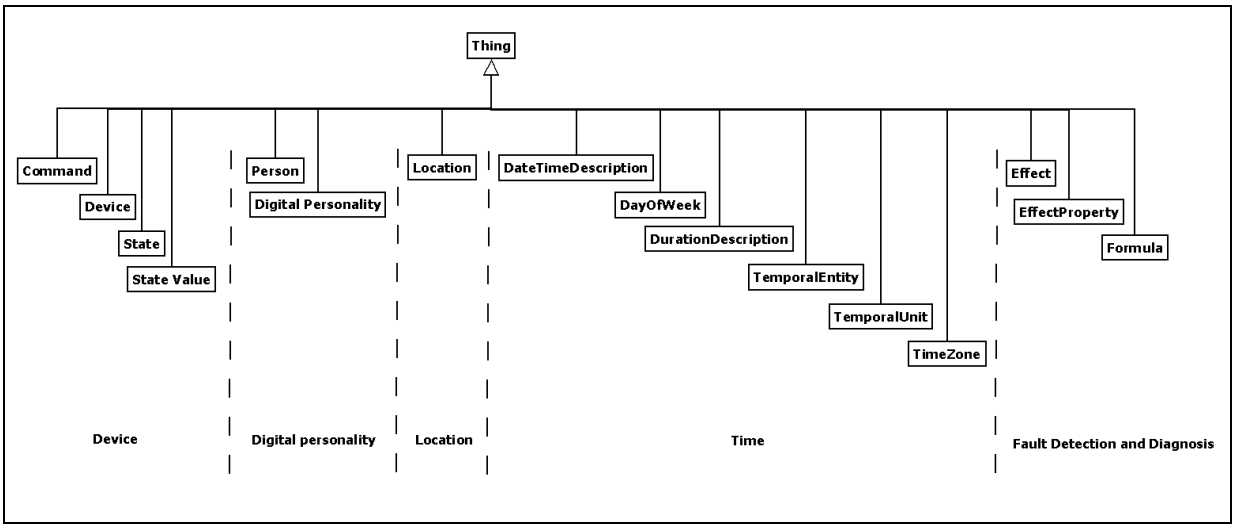

Figure 74. First level of the CBDP ontology

The fact that the ontology is loosely coupled with the framework makes changing the ontology without affecting the framework easier. This fact facilitates integrating our main FDD framework entities into the CBDP framework. However some basic parts of the ontology are fixed and cannot be changed, for instance the entities describing the delivery of commands to the physical devices. The entities describing this functionality are depicted in **Figure 75**.

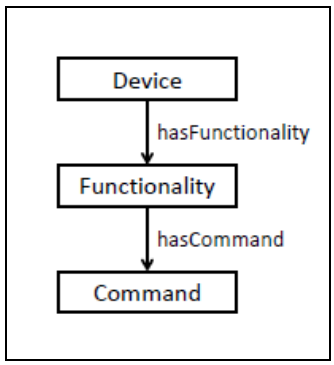

Figure 75. Ontology entities required for proper operation of the CBDP framework

5.1.2. The CBDP framework

The CBDP Framework aims to dynamically handle ontology data in order to initiate actions when specified conditions in the ontology are verified. The CBDP framework is written in Java and it is based on OSGi. OSGi allows one to flexibly build applications by combining bundles. The physical devices (actuators, sensors) are connected to the platform using the Zigbee wireless communications protocol.

As described in **Figure 76** a typical CBDP application is composed of CBDP's core bundles, which are the Context Reasoner and the Sensor/Actuator Layer, and application-specific bundles.

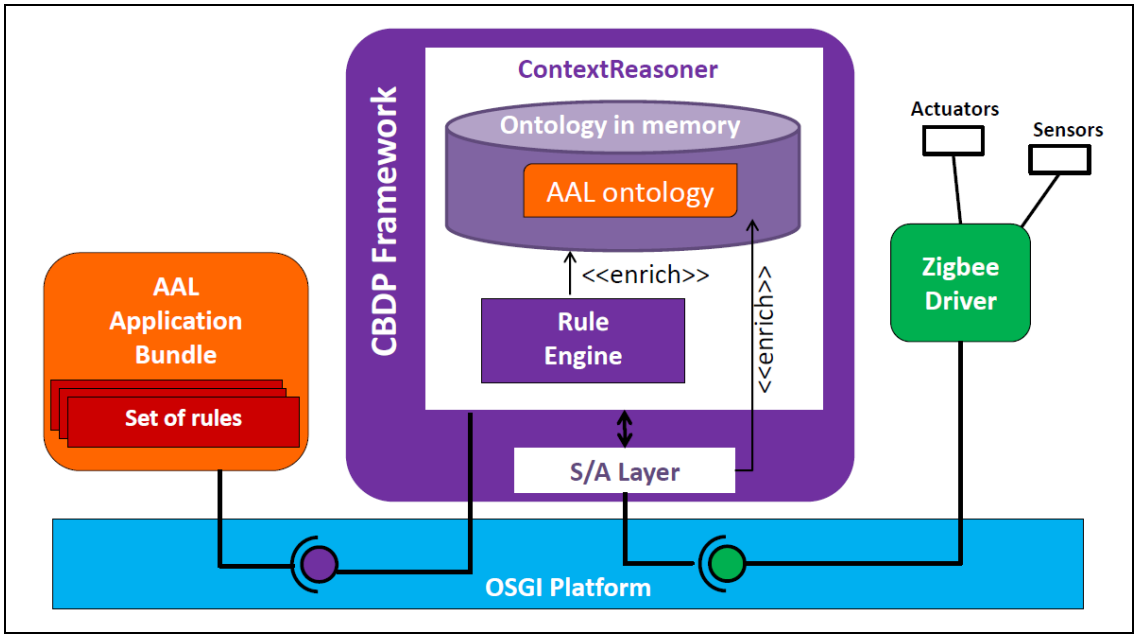

Figure 76. Architecture of the CBDP framework

- **Context Reasoner**: it manages (add, retrieve, perform queries about) the information coming from external components, such as the AAL Application or the Zigbee Driver by structuring them according to the used ontology. The ontology is manipulated using the Jena library **[193]**. The Context Reasoner has a rule engine. The purpose of the latter is to select the appropriate actions to perform to help the user and facilitate common tasks, based on a set of application-specific rules, which is why the rules are provided by a bundle specific to the application (AAL Application bundle in our case). The rules are in the form of Horn clauses **[194]**. This means that a rule is composed of two main parts: the premises that determine the conditions under

which the rule applies, and a conclusion that basically adds a new "fact" into the ontology (such as a new property value). An example of such rules in pseudo code is the following:

```
IF a PresenceSensor detects somebody
AND the PresenceSensor,LightActuator are in the same room
      THEN Turn the LightActuator on
```

The exact rule expressed in Jena syntax is at **Annex-D.Rule1**. This rule is evaluated by the framework when the Presence Sensor's value has changed in the ontology.

Rules like this are applied by Jena's basic reasoning engine, using forward chaining. In order to avoid applying all the rules at each instant, only rules matching some application-specific filters are applied. These filters are defined by the bundle of the application-specific bundle. This idea solves performance issues, which is why a "catch-all" filter is used at first, and then the filters are refined in a way that enhances the performance.

- **Sensor/Actuator Layer**: it connects the sensors and actuators to the ontology. The communication goes in two directions: The *first direction* is to send Sensor data (through Zigbee) to be stored in the ontology. This allows the performing of semantic queries and semantic reasoning over sensor data. The second direction is when command requests are inserted in the ontology (using a property called hasCommand), which triggers the actual emission of a command to the actuator. A specific OSGi service called EventAdmin is responsible for connecting the sensors to the Context Reasoner. In order to allow the communication between the drivers and the S/A layer, a specific communication protocol has been defined through OSGi events. A description of this communication protocol is given in **[195]**.
- **Application-specific bundles for the Ambient Assisted Living application**: In the case of the AAL application there are two main bundles, which are:

  (*i*) *The AAL-specific application bundle* that contains the rules that define the wanted application behavior, which are meant to assist the user according to his/her needs.

  (*ii*) *The Zigbee Driver bundle*, which allows exchanging data between the physical devices that are connected via a wireless Zigbee network, and the CBDP Framework.

5.1.3. Integration of the Fault Detection and Diagnosis approach with the CBDP framework

This section shows how our FDD framework may be integrated with CBDP. As shown in **Figure 77**, the main concepts from our FDD framework can be integrated into the CBDP ontology, namely: Effect, Effect Property, and Formula.

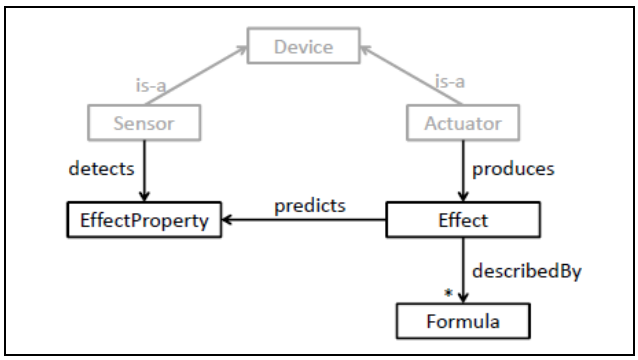

Figure 77. The Fault Detection and Diagnosis framework integration into the CBDP ontology (in grey) – Abstract Model

This results in a similar integration at the level of the Concrete Model, as shown in **Figure 78**:

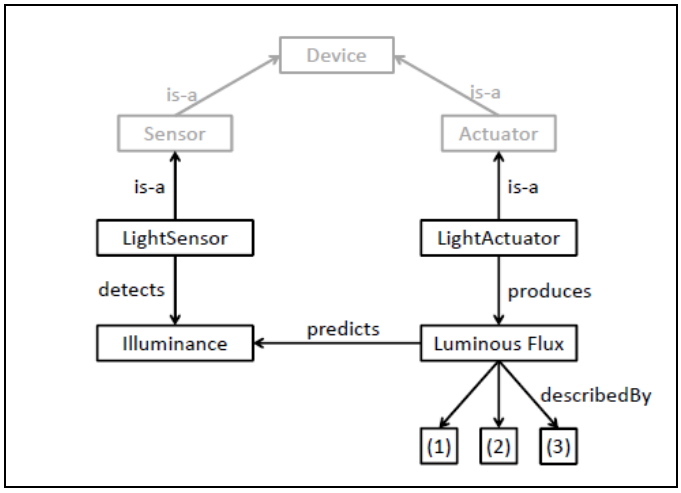

Figure 78. Specialization for the Light Effect from the CBDP ontology – Concrete Model

5.2. ModHel'X, our heterogeneous modeling tool

As explained in introduction, the prediction model is heterogeneous by nature, because it contains, namely, state machines and computations. Instead of creating an ad-hoc execution engine for this model, we have decided to use a specialized tool. We have chosen to use ModHel'X, which is developed at Supélec, and that has an experimental, yet clean and usable implementation.

The goal of ModHel'X **[196][172]** is to develop new ideas about the executable semantics of heterogeneous models. There are two main tasks to achieve in order to obtain a meaningful heterogeneous model using model composition:

(1) the precise definition of the semantics of each modeling language;

(2) the precise definition of the semantic adaptation between parts of a model that use different modeling languages.

In order to define the semantics of different modeling languages, ModHel'X is composed of a generic meta-model for describing the structure of models, and a generic execution engine for interpreting such structures. To attach semantics to this structure, ModHel'X uses the concept of model of computation (MoC). The model of computation in ModHel'X handles the scheduling of actions for the models' components, and how values are transferred between different model components **[196]**, thereby refining the execution engine's abstract semantics. ModHel'X currently implements three MoCs: timed finite state machines (TFSM), discrete events (DE), synchronous data flow (SDF). Refer to Section 3.4 for a description of these common MoCs. Those who create models may use these MoCs off-the-shelf.

For instance, **Figure 79** shows that two models can share the same structure (two components A and B linked by two arrows) with different semantics, i.e., different MoC: a finite state machine (FSM) or two processes communicating through discrete events (DE). When interpreted by the FSM MoC, the model represents an FSM with two states. When interpreted by the DE MoC, it represents two processes that communicate through events.

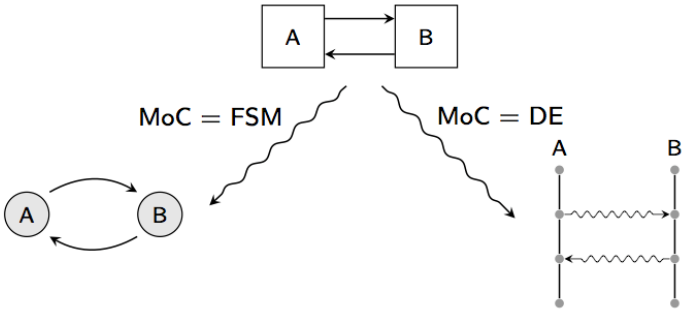

Figure 79. A model (top) that can be interpreted according to two different MoCs (bottom)

The elementary unit of behavior in ModHel'X is the block, which is composed, as shown in **Figure 80**, of an interface (pins) and of an update operation allowing to observe the behavior of the block through the interface. A block is a black box: the pins are responsible of sending and receiving information and defining what can be observed from the outside of the block. To observe the block's behavior the update operation of the block's interface is invoked, causing the block to consider its input and update the output according to inputs and current state of the block.

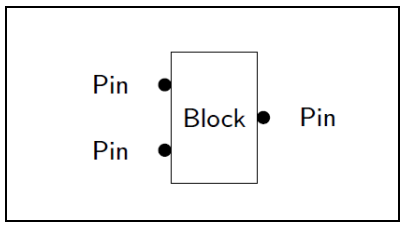

Figure 80. A ModHel'X block

As shown on **Figure 81**, a model is composed of a structure, which contains blocks interconnected via relations (arrows), and which is interpreted according to a MoC (shown in a diamond-shaped label). Interpreting a model means executing the behavior described by that model according to the semantics of the MoC. An execution is a series of observations of the model, each observation being computed through the sequential observation of the blocks of the model using a fixed-point algorithm. The observation of one block is called an update. Each MoC dictates the rules for scheduling the update of the blocks of a model, for propagating values between blocks, and for determining when the computation of the observation of the model is complete.

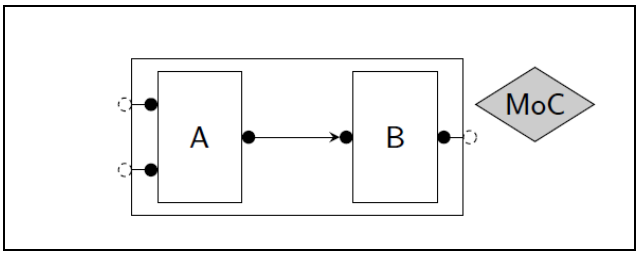

Figure 81. A ModHel'X model

For hierarchical composition and heterogeneity support ModHel'X uses the concept of interface block, which is a block whose behavior is defined by a model inside it, as shown in **Figure 82**. The MoC used by the inner model of an interface block can differ from the MoC of the outer model to which the interface block belongs. The InterfaceBlock acts as an adapter

between the two MoCs. The dashed arrows between the pins of the interface block and the pins of the inner model represent the semantic adaptation between the two MoCs, which is realized by the interface block. As shown in **[172]**, three aspects are be considered in this adaptation: data (which may not have the same form in the inner and outer models), time (the notion of time and the time scales may differ in the inner and outer models) and control (the instants at which it is possible or necessary to communicate with a block through its interface).

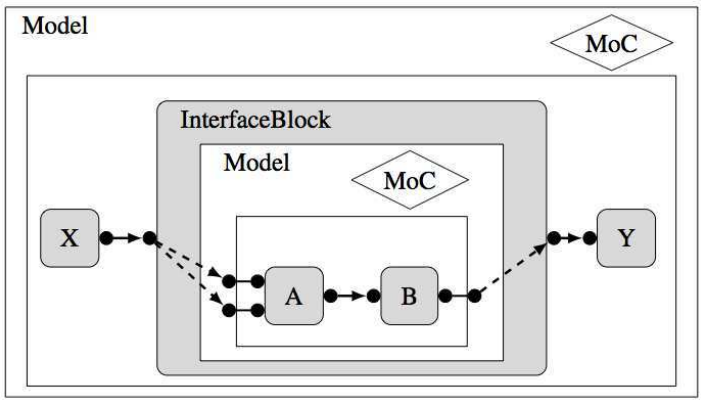

Figure 82. A ModHel'X interface block

ModHel'X allows the creation of models through an API, and offers a graphical animator for visualizing models. It executes models in simulated time or in real-time. Therefore ModHel'X can be used both for performing simulations or running actual systems. For instance it has been used to manage the interaction between a user and a virtual environment, the gestures of the user being captured by a Kinect device **[197]**.

Therefore, the core of the work of the FDD framework is to build a prediction model using the API provided by ModHel'X; after that ModHel'X can execute this model autonomously.

5.3. Building the prediction model

We have implemented our FDD framework using Java. It creates a prediction model to be executed by ModHel'X, as a simulation or in real-scale. Using this configuration we were able to define several scenarios, and to test our Fault Detection approach results in those scenarios (see **Chapter 6**).

In input the Java application takes the definition of the Concrete Model and the Instances, and in output it uses the API of ModHel'X to create the Prediction Model is depicted in **Figure 83**.

The complete description of the Ambient System to be diagnosed by the FDD framework is described in the form of triplet statements *(Models.alpl)*. The latter file contains two main sections, the *class* section, which has the definition of the types (the concrete model), and an *instance* section that holds the instances corresponding to the real devices and objects in the ambient intelligent environment that are involved in the fault detection and diagnosis process (see next paragraph **5.3.1** for detailed syntax).

For some types of devices described in the *class* section there is a reference to behavioral models (Finite State Machines in our case) described in xml files (*FSM.xml*).

The models in *Models.alpl* file are mirrored in an OWL ontology (*Ontology.owl*). In addition for being the medium for the integration with the CBDP project, the ontology contains the Abstract Model description; hence it is used for analyzing the Concrete Models described in the *Models.alpl* file.

The analysis process can be summerized in two main tasks:

- Verifying that the abstract types used in the Concrete Model exist in the Abstract Model.

- Verifying the correctness of the structure, this means verifying that the links that are defined between the concrete types are also defined between the abstract types from which the concrete types are inherited.

We use the Jena library **[193]** to manipulate the ontology, in particular to create ontology entities that mirror the classes and devices' instances described in the *Models.alpl* file, and/or instances that are added at run-time.

All this information is used to generate a Prediction Model in ModHel'X.

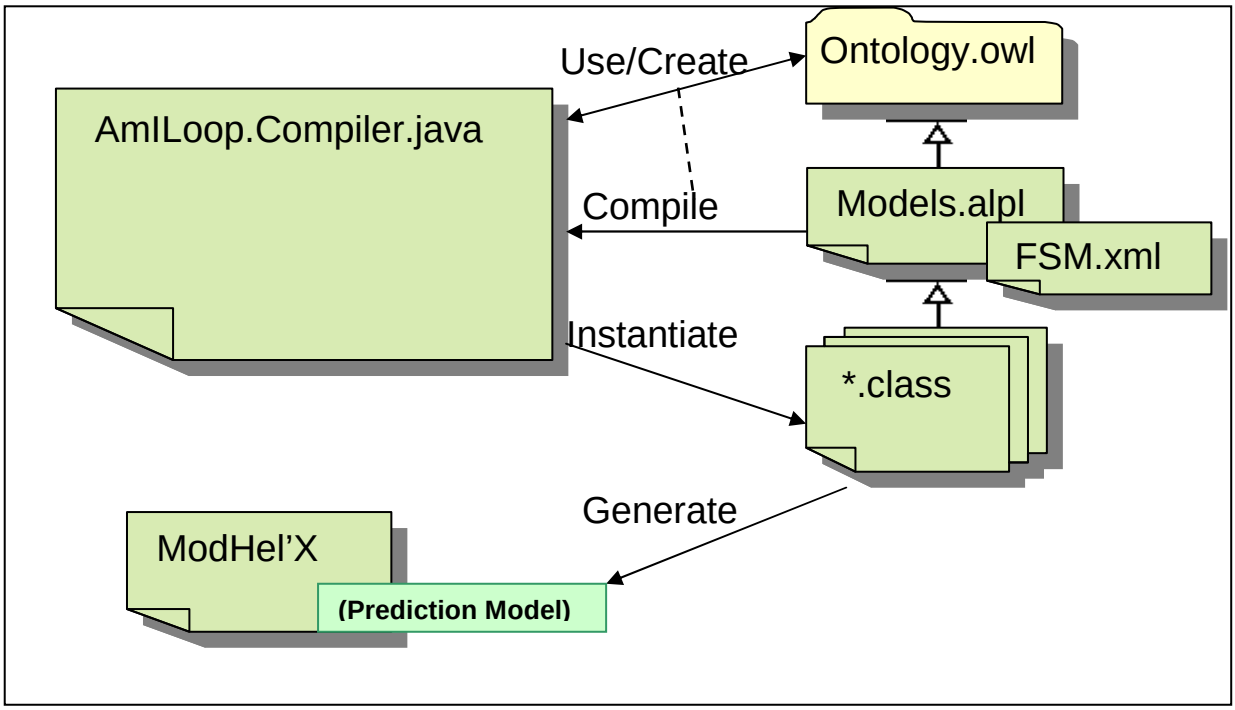

Figure 83. File structure of the java application

5.3.1. Defining the Concrete Model and the Instances of the actual devices

In the various files, our models (the concrete model and the instances) are described in the form of statements of the form:

```
subject-predicate-object
```

This is inspired by Resource Description Framework (RDF) **[190]**.

The subject in this case is the resource to be described, the object is usually another resource or a literal value, and the predicate denotes the relationship between the subject and the object.

For example, one way for declaring the fact that "a light bulb is producing light effect, and the light effect has a property called luminous flux", using triples is in fact composed of two statements:

```
light_bulb_1 produce light_effect
light_effect has_property luminous_flux
```

In the latter statements we have a triplet composed of:

entity-relationship-entity

Note here that the relationship names are as declared in the abstract model (**Figure 48**).

Continuing with the same example, to initialize the value of the luminous flux of the light effect to a certain value (100 lumens for example), we simply state:

```
luminous_flux has_value 100
```

which is a statement composed of:

entity-attribute-value

Using this syntax we have created our own grammar to describe the all models composing the fault detection and diagnosis framework. In addition to the relationships between entities that are already declared in the abstract model, we have added some relationships that describe some implicit relationships in the models such as the inheritance and the assignment of a value to an entity. For the latter we added the relationship ***has_value*** to be used as in the previous example.

Out *Models.alpl* file is composed of two main sections:

- The Classes section: containing the declaration of the concrete model entities from the abstract model.

To declare an entity of the concrete model that is a specialization of an entity of the abstract model we use the keyword: ***TYPE***. For instance to declare the type of incandescent light bulb, which is a type of actuators we use the triplet:

```
IncandescentLightBulb TYPE Actuator
```

- The Instances section: containing the declaration of the instances from the types defined in the concrete Model (the Classes section).

To declare an instance an entity declared in the concrete model we use the keyword: ***is***. For example to create two instances of incandescent light bulb we use the two triples:

```
Bulb_1 is IncandescentLightBulb
Bulb_2 is IncandescentLightBulb
```

5.3.2. Defining the Behavioral Models and their instantiation

The generic behavioral models of the active types (Actuators and Effect Modifiers) are defined in separate XML files. The latter are associated in the Concrete Model to their declared types with their file paths:

```
FluorescentLightBulb FSM resources/FSM/FluorescentLightBulb_FSM.xml;
```

For instance the structure of the FSM defined in **Figure 68** for Fluorescent Light Bulbs in XML format would be:

```
<FSM Name="FluorescentLightBulb_FSM" ControlledPropertyName="LightFlux">
   <EventInput Name='TurnON' />
   <EventInput Name='TurnOFF' />

   <Output Name='LightFlux' />


   <State Name="OFF" Value="0">
         <Event Name="TurnON" NextState="HEATING">
               <Set Output="LightFlux" Value="50" />
```

```
        </Event>
    </State>

    <State Name="HEATING" Value="50">
        <Event Delay="30" NextState="ON">
            <Set Output="LightFlux" Value="1500" />
        </Event>
        <Event Name="TurnOFF" NextState="OFF">
            <Set Output="LightFlux" Value="0" />
        </Event>
    </State>

    <State Name="ON" Value="1500">
        <Event Name="TurnOFF" NextState="COOLING">
            <Set Output="LightFlux" Value="0" />
        </Event>
    </State>

    <State Name="COOLING" Value="0">
        <Event Delay="60" NextState="OFF">
            <Set Output="LightFlux" Value="0" />
        </Event>
        <Event Name="TurnON" NextState="ON">
            <Set Output="LightFlux" Value="1500" />
        </Event>
    </State>

</FSM>
```

The algorithm that generates the Prediction engine will instantiate these Behavioral Models for each instance and replace the variables with their values defined for each Instance in the Instances section. It will then convert them to ModHel'X blocks as the one depicted in **Figure 84**:

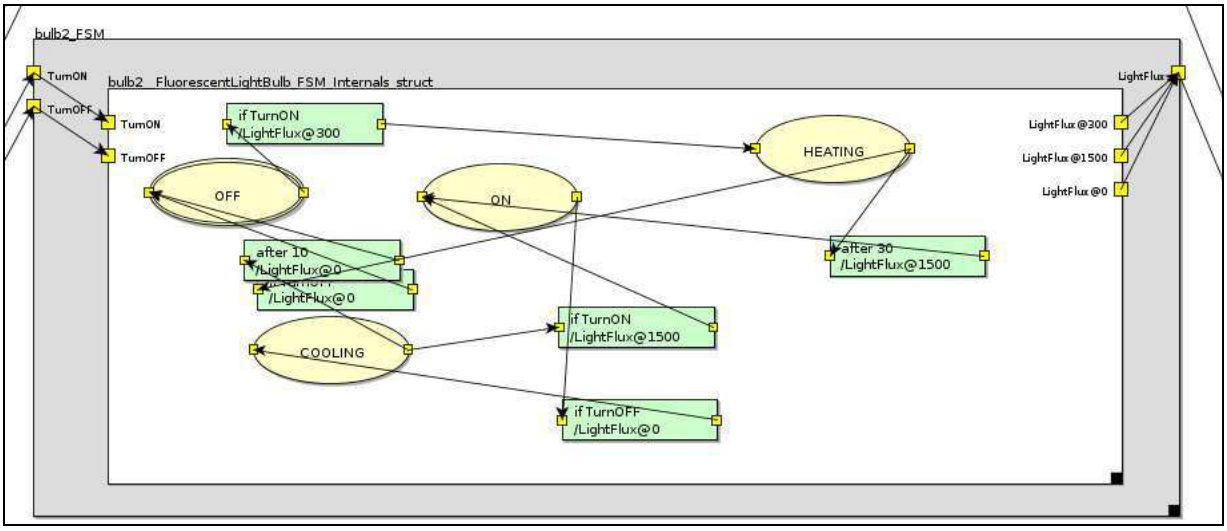

Figure 84. ModHel'X block describing CFL Bulb FSM

5.3.3. Generating the Prediction Model (Prediction Engine)

Based on the previous declarations, the Prediction Engine will generate the Prediction Model. The Prediction Model must be regenerated each time the *structure* of the environment changes, namely when a new object is introduced in the ambient environment, and each time an object is

removed. However when it the case of variations of property values, it is not necessary to regenerate the model because the *structure* does not change; we must only continue its execution with new input values.

We must determine the expected sensor outputs using the formulae associated with the effects. Starting from the needed values, we recursively iterate over the formulae and we construct symbolic call tree. For instance the call tree for a single light sensor expanding the (L4) law from the *2DLightLawSet* (see **4.2.1.2.2** for the complete Law-Set) and two light sources would result in the call tree depicted on **Figure 85**. In the process, iterating operators, such as the sigma notation for a sum, must be expanded depending on the current situation of the environment. For instance, when dealing with a summation operator that iterates over "all the light sources", we actually iterate over the light sources currently present in the environment and create as many branches in the call tree as there are light sources. Therefore the current situation of the environment determines the structure of the call tree, hence the structure of the model.

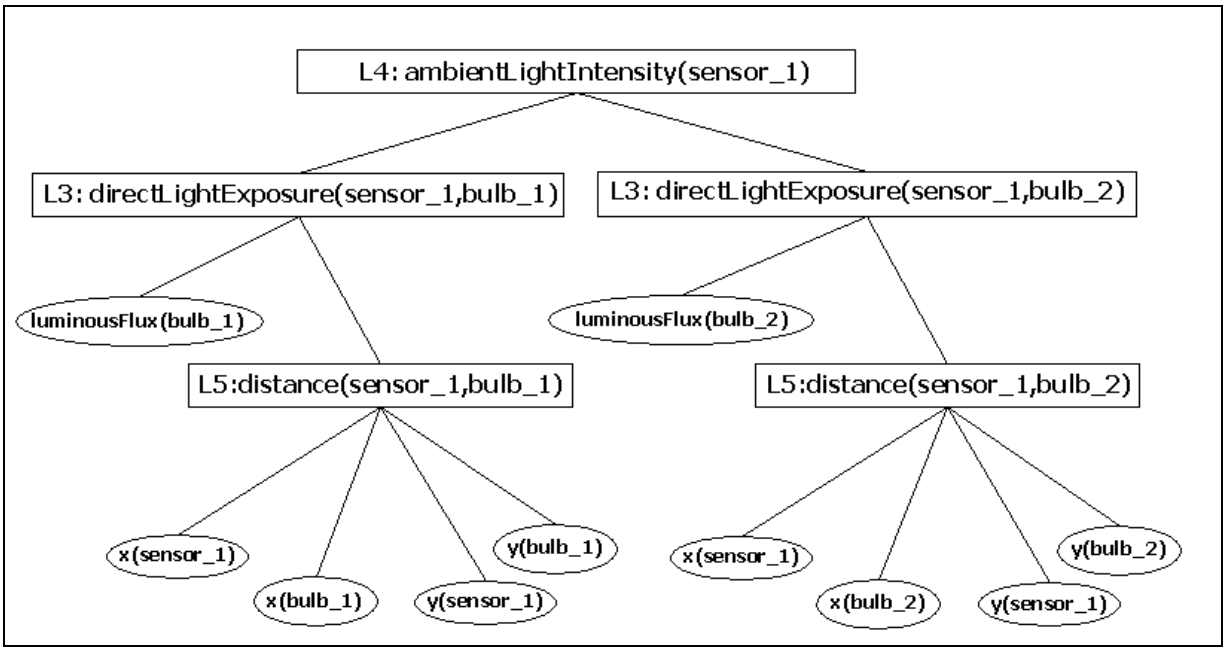

Figure 85. Call tree for a single sensor in the Prediction Model

This call tree is directly translated into a dataflow model, with each elementary function being an operator. Depending on the library of operators available, an operator can yield a single block in ModHel'X, or a sub-tree of lower-level blocks. The Model of Computation chosen for this computational model is Synchronous Dataflow (SDF). SDF enables us to treat the computational model as a sample system, computing its new outputs regularly, for instance each second. **Figure 86** shows (the highlighted area) the dataflow model associated with one call tree that has three main branches (predicting the reading of a sensor that is exposed to three light sources).

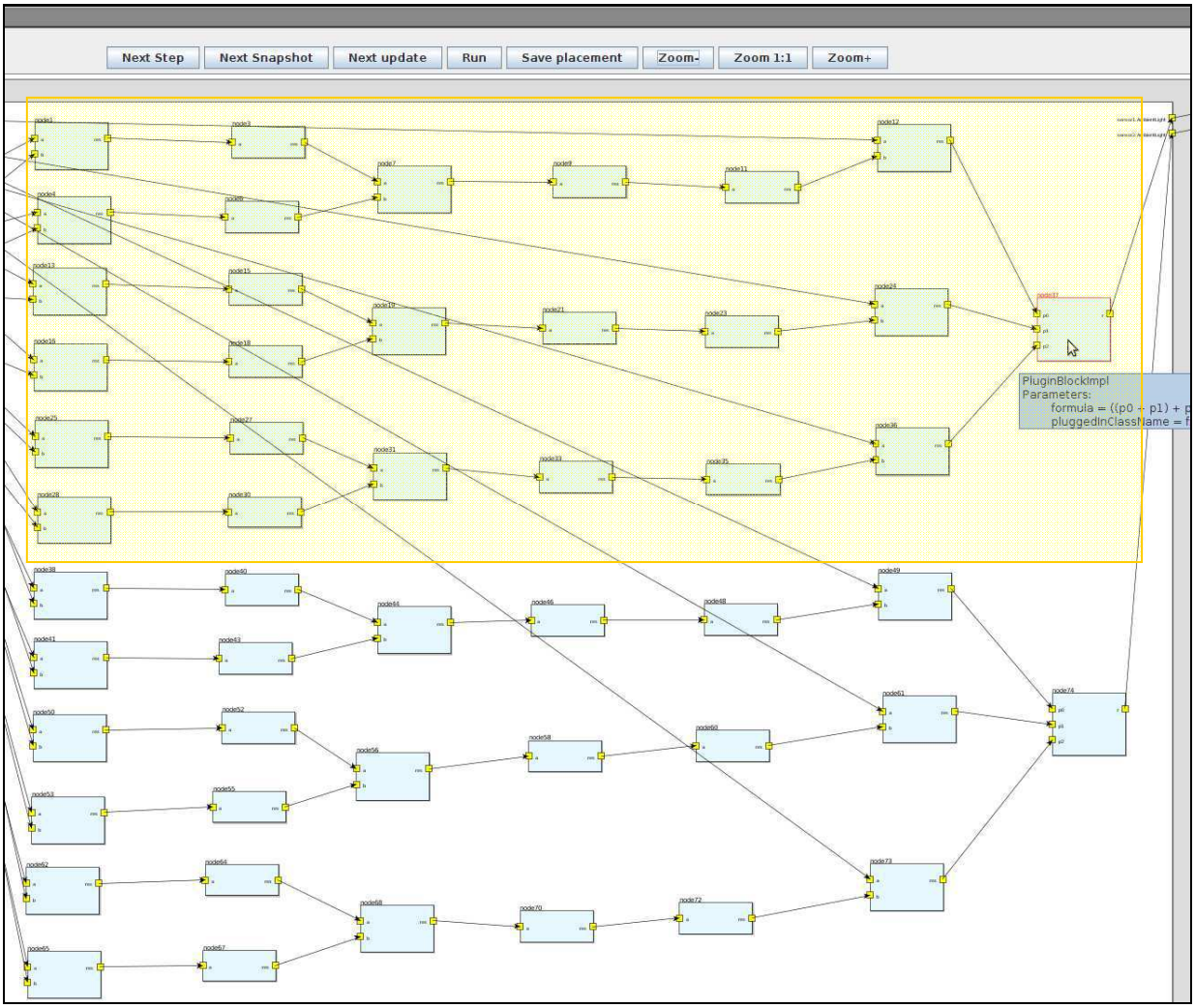

Figure 86. Dataflow model in ModHel'X - A Single Call tree highlighted

This output of some actuators depends on their behavioral model. For instance, the luminous flux of a light bulb depends on the state it is currently in (switched off, heating or switched on). Therefore the state machines must be integrated in the prediction model. In this way, the prediction model can not only be used to compute new predictions based on new values, but also taking into account the flow of commands sent to the system (for example, "switch on bulb_1", "switch off bulb_2", etc.). The state machines are modeled using the TFSM Model of Computation. Using timed state machines is necessary, because some transitions fire spontaneously after some time (for instance, the state machine modeling a light bulb might transition from "heating" to "switched on" after 30 seconds in the former state.

At this point we have two parts in the model: a computational dataflow model using the SDF MoC, and a number of state machines using the TFSM MoC. Of course, the dataflow calculations depend on the current states of the state machines. Therefore the two heterogeneous parts must be integrated. Therefore a top-level model using the discrete events (DE) MoC is created. Discrete events allow state changes to be propagated to the SDF model. The TFSMs and the SDF model are plunged into this DE model, and semantic adaptation is performed using adapters (interface blocks).

At the boundary between DE and TFSM, in input, events such as "switch on this lamp" just have to be forwarded to the TFSM. In output, the TFSM produces events such as "Light flux now at 1500 lm" that also just have to be forwarded to DE.

The adaptation is more complex at the boundary between DE and SDF. In input, and event such as "Light flux now at 1500 lm" cannot in general be taken into account immediately. Indeed, an SDF model is a sampled system that reacts at a specific rate. Therefore, events are translated into values, and these values are memorized so as to be provided to the model at its next activation instant. For example, "Light flux now at 1500 lm" creates a new value of 1500 on the pin corresponding to the actuator's light flux, and after receiving this event, this new value is emitted at each SDF instant. Therefore, the DE/SDF adapter embeds some kind of "translation table". A generic DE/SDF adapter is provided by ModHel'X, that can be parameterized with such a custom translation table. In output, the adapter sends an event in DE only when a value changes in SDF, so as not to create too many events.

5.4. Execution of the Prediction Model

The Prediction Model may be used for simulation or for running an actual system. It is just a matter of connecting either simulation components or actual sensors in input of the model.

At the moment, we have only performed tests using simulation. Therefore we have built a control panel to generate all kinds of events, such as commands sent to the actuators, or values read from the sensors.

5.4.1. Use of the Prediction Model in simulation

When the Prediction Model is run, ModHel'X provides us with an animator that may be used to understand and debug the model. We obtain an interface like the one depicted in **Figure 87** :

(a.) The ModHel'X main block inputs: It simulates the data coming from the hardware layer. **Figure 88** is a zoom in on this part.

(b.) The finite state machines (**Figure 84** is a zoom-in on one of the finite state machines).

(c.) The calculations block representing the call trees generated of the Law Sets (mathematical model)

(d.) The Prediction Model output: or the results of the calculations. It represents the sensors' readings predicted values. This part is zoomed in on **Figure 89**. Note that we have as many outputs as the number of sensors in the environment.

We can start the simulation by clicking run. One second is the time step. Once the simulation is running we have two new graphic interfaces:

i) One is for controlling the values for the properties of the instances (for instance x and y values), and for triggering the transitions in the finite state machines (for instance turning on and off the bulbs), as showed in **Figure 90**.

With this interface we can simulate the behavior of the real devices of the environment by changing the values of their properties.

ii) The second graphic interface is for tracing the Prediction Model output, which in this case is the sensors' predicted readings values, as showed in **Figure 91**.

The Prediction Model output is in fact the result of the applicable law-set for each sensor present in the environment.

There is also a log file to save the history of the predicted values, and the graphic layout of the Prediction Model can also be saved.

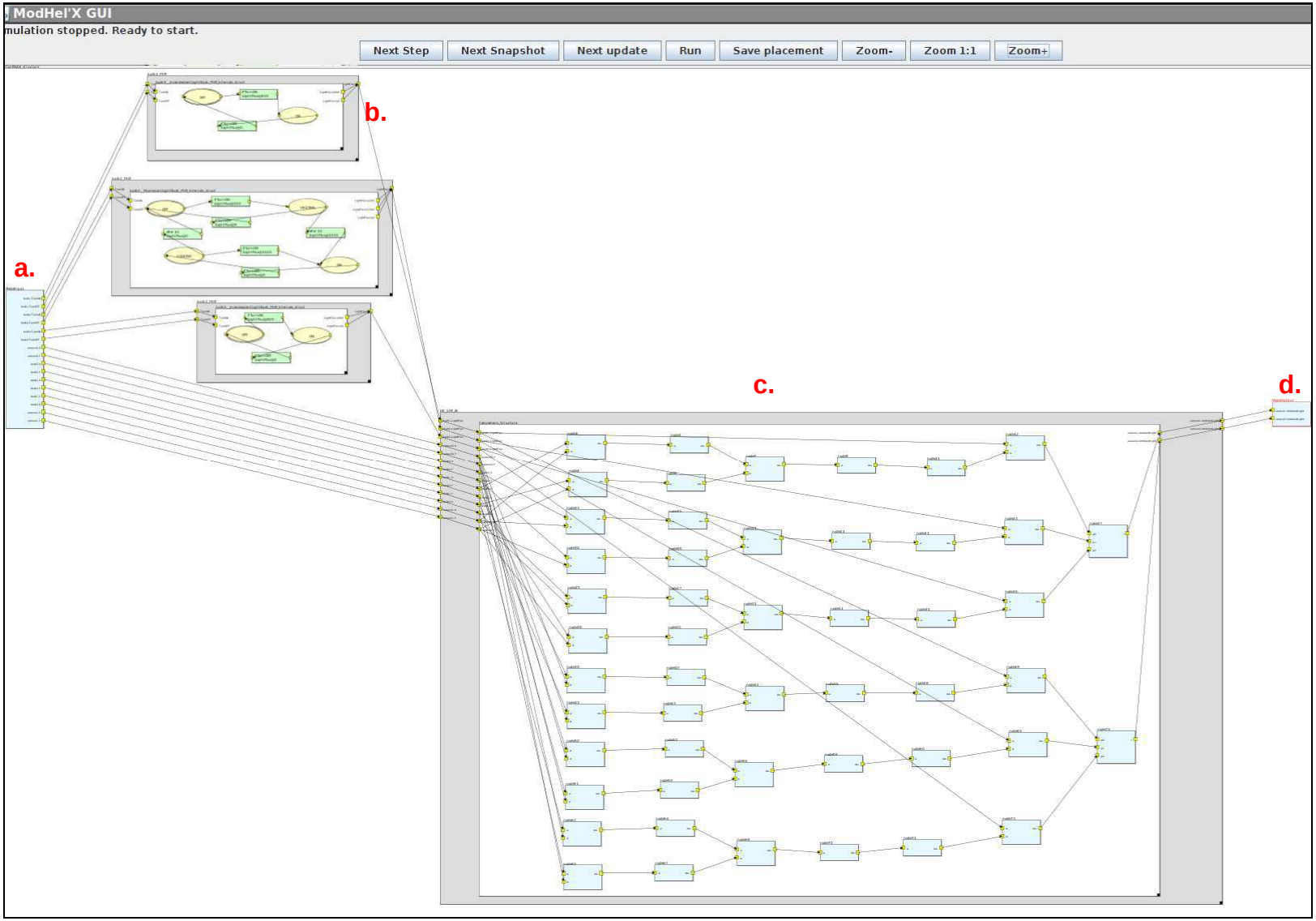

Figure 87. A Zoomed-out view of the ModHel'X representation of the Prediction Model

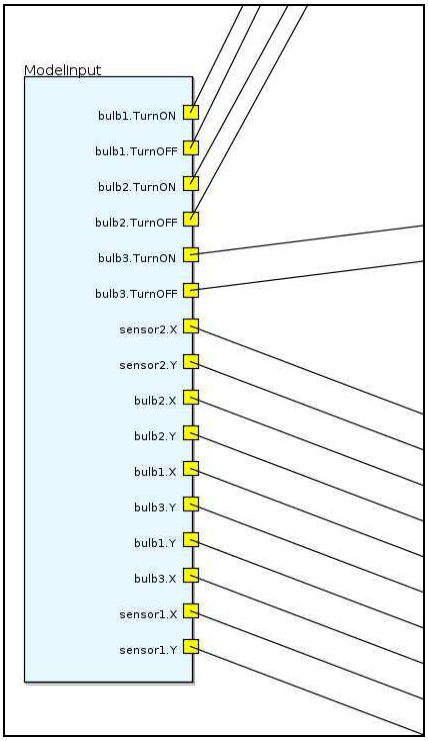

Figure 88. The ModHel'X main block inputs

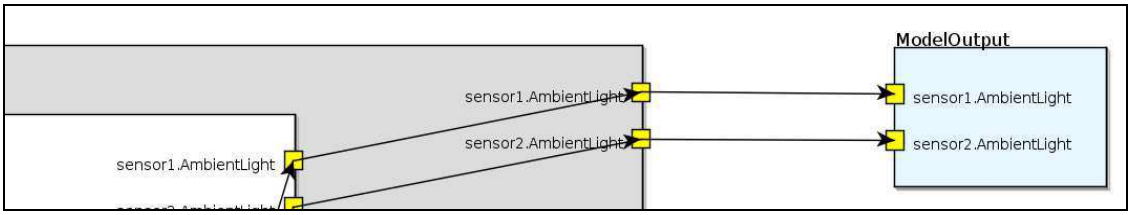

Figure 89. The Prediction Model outputs

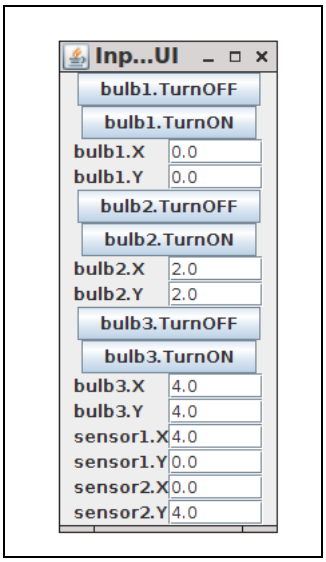

Figure 90. Control panel for the Prediction Model inputs

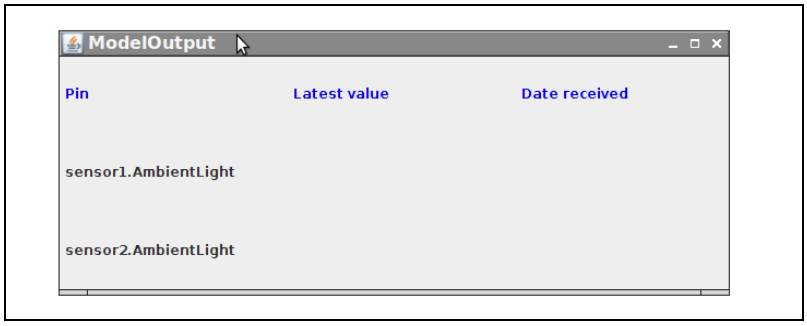

Figure 91. Prediction Model's output trace window

5.4.2. Fault Detection

Once the execution of the Prediction Model provides values expected to be read for each sensor, fault detection is a trivial task. One has just to compare the expected values with the actual values. If they differ beyond a specified tolerance, then there is a fault.

Next we propose a generic fault detection algorithm in pseudo-code. The idea is to loop over all sensors in the Prediction Model and compare, for each one of them, their actual readings with the predicted value by the Prediction Model.

```
FOR EACH Sensor in the PredictionModel
   IF Sensor.reading is NOT IN
         [PredictionModel.PredictedValue(Sensor)-Sensor.tolerance,
         PredictionModel.PredictedValue(Sensor)+Sensor.tolerance]
   THEN
         Sensor.info="Fault Detected"
   END IF
END FOR EACH
```

Note how the tolerance value is considered when comparing the theoretical and the actual reading. The tolerance value can be defined as a property of the Sensor or it can be deduced from the accuracy value (the difference between the actual value and the value that can be read from the output of the sensor), which is usually a characteristic of the sensor given by their constructors.

5.5. Conclusion

In this chapter we have proposed an implementation of our Fault Detection and Diagnosis framework that can be integrated into an actual AAL framework, the CBDP framework. The main task of the FDD framework is to generate a Prediction Model, of heterogeneous nature, that can be executed by a dedicated tool called ModHel'X. The results of the execution of the model enable one to actually detect faults.

Currently the execution has been performed in a simulated environment. However, with minor changes we could use exactly the same toolchain to diagnose a real-scale AAL application.

In the next chapter we detail scenarios that were used to test our framework.

Chapter 6:

Application Examples of Fault Detection and Diagnosis in a Smart Home

Chapter 6.

Application Examples of Fault Detection and Diagnosis in a Smart Home

In this chapter we present four complete examples, in which our FDD framework is used to perform fault detection and diagnosis on a smart environment. The first example is a scenario that was run in an AAL application setting in the context of the CBDP European project validation (we call it **Scenario_CBDP**). The rest of the examples (we Call **Scenario_1**, **Scenario_2**, and **Scenario_3**) are scenarios imagined and simulated using our java simulator presented in the previous chapter. For our simulator scenarios we suppose that we have three main Ambient Systems running in our environment: Ambient Light System, Ambient Heating System, and Ambient Hot-Bath System.

For simplification reasons (not to have crowded illustrations of the design models) fault detection and diagnosis of different systems will be described separately. **Scenario_1** and **Scenario_2** will describe the fault detection in an Ambient Light System, and **Scenario_3** will describe fault detection of Heating and Water Flow in an Ambient Hot-Bath System.

For the first example (**scenario_CBDP**), we focus on "when and how" our Fault Detection and Diagnosis framework intervenes in order to contribute to ameliorating the overall execution of an Ambient Intelligent System. So, for this scenario, we describe a general AAL scenario and we highlight the parts where our FDD is executed.

For the rest of the examples, we focus on showing the "step by step building" of the FDD framework. So, for each example, we start by a description of the Ambient Intelligent Environment, then we start building the Models of our FDD framework, starting with the Concrete Model, by defining the types of the devices in the environment and the physical phenomena that are going to be observed in the environment, then we fix the Mathematical Model by choosing the appropriate Law-Sets that are going to be used by the Prediction Engine, then we define our Behavioral Models for the different types of active devices, then we instantiate the Models with the actual devices, finally we run a simulation to perform fault detection using ModHel'X and we compare the results with the theoretical values to validate the framework.

6.1. Scenario_CBDP: Automatic Light Switch with Light System fault detection and diagnosis

6.1.1. Description of the ambient environment

We suppose that we have a room composed of:

- A controlled light bulb
- A light sensor
- A presence sensor
- A non controlled (human-operated) lamp: even though this light source is not controlled, the position of the adjustment knob allowing the user to manually control the intensity of the light is known; hence the system can know what light flux is generated from this source at any time.

As shown in **Figure 92**, the input and output devices used in this scenario are all Zigbee devices controlled by the Zigbee driver (see **3.2.1**), which connects to the CBDP platform through OSGi **[188]**. The light actuators and the light sensors are all in the same room, so naturally the light sensor detects light emitted by both light bulbs, the controlled one and the human-operated one.

The functionality detailed in this Ambient Assisted Living scenario aims to help elderly people avoid finding themselves lost in a dark room. The wanted system behavior may be summarized by this rule:

"**if the ambient light level is under a threshold of a specific user (specified in the Digital Personality) and if the user is present in the room, then the light must be turned on**".

Although simple, this scenario demonstrates all the aspects of the system: *sensor data gathering*, *reasoning*, *command of actuators*, and *fault detection and diagnosis*.

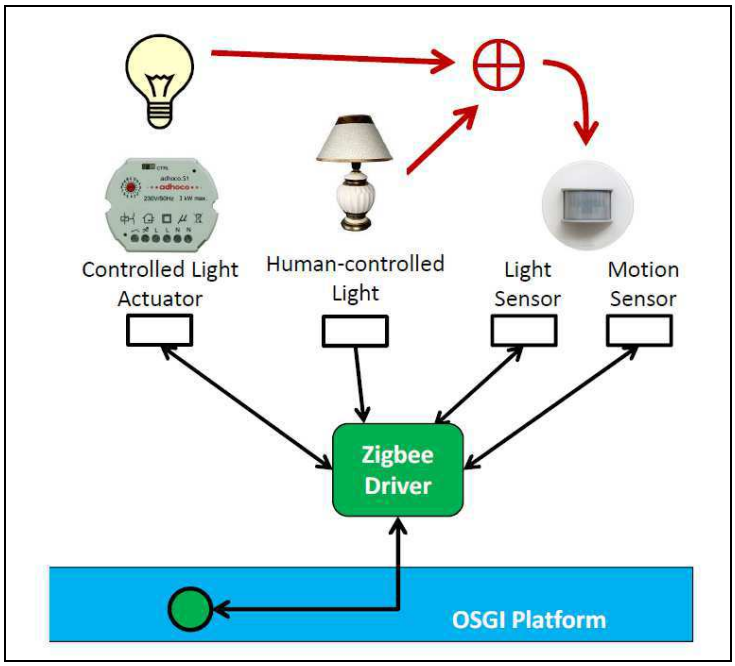

Figure 92. Input/Output configuration of scenario_CBDP

6.1.2. Running the scenario

In this scenario we suppose that the Illuminance in the room is 80 lux coming the the light flux of the human-operated lamp. The current Illuminance value is updated in the ontology (see **5.1** for a complete description of the CBDP framework). Then a user comes in the room. The system verifies his/her Digital Personality, which states that he/she does not like to be in a room where the Illuminance is under 100 lux. The system then launches the following sequence of actions:

1) When the user enters the room, the PresenceSensor sends a notification to the driver through Zigbee network. The driver sends an event to the framework and the ontology is then updated to consider the new change in the user's position.

2) The framework detects this change in the value of the PresenceSensor and evaluates the following Jena rule, expressed here in pseudo-natural language for simplification reasons (for the exact rule in Jena syntax see **Annex-D.Rule2**):

```
IF a LightSensor value is <{userPreference in the Digital Personality}
AND a PresenceSensor detects somebody
AND the LightSensor,PresenceSensor,LightActuator are in the same room
      THEN Turn the LightActuator on
```

The reasoning engine evaluates the rule by reading the current light level, the current presence status, and the user preferences from the ontology. The premises of the rules are evaluated as true, so the conclusion is by consequence executed. To determine the rules to apply, forward chaining reasoning algorithm is used by Jena.

3) This rule causes the adding of a new statement to the ontology, which is:

```
LightActuator hasCommand OnCommand
```

The Sensor/Actuator Layer (as described in **5.1**) detects the predicate **`hasCommand`**. In consequence, the framework sends an event to the driver asking for the Light Actuator to be turn on.

4) Through Zigbee network, the driver commands the Light Actuator, which turns the light on.

5) It is at this point that the system calls our Fault Detection and Diagnosis components to determine whether the action described in 4) was successfully executed.

In this scenario we have an Ambient Environment equipped with two light actuators (we'll call their two instances **`la1`** and **`la2`** respectively), and a light sensor (**`ls1`**). The light actuators are instances of the concrete type *LightActuator*, and the light sensor is instance of the concrete type *LightSensor* (see **Figure 78** for the Concrete Model). Note that initially there is absolutely no link between the actuators and the sensor. The links will be deduced automatically at run-time via the Prediction Model. This is done when the mathematical formula's input variables are replaced with the properties of actual instances and the iterative functions are expanded. So the mathematical function **(3)** in **5.1.3** becomes:

$$\texttt{ls1.illuminance} = \texttt{illum}(\texttt{la1}, \texttt{ls1}) + \texttt{illum}(\texttt{la2}, \texttt{ls2})$$

Replacing **`illum`** functions with their respective expressions gives:

$$\texttt{ls1.illuminance} = \frac{\texttt{la1.flux}}{(\texttt{distance}(\texttt{la1}, \texttt{ls1}))^2} + \frac{\texttt{la2.flux}}{(\texttt{distance}(\texttt{la2}, \texttt{ls1}))^2}$$

Then we replace **`distance`** functions with their expressions:

$$\texttt{ls1.illuminance} = \frac{\texttt{la1.flux}}{(\texttt{la1.x} - \texttt{ls1.x})^2 + (\texttt{la1.y} - \texttt{ls1.y})^2} + \frac{\texttt{la2.flux}}{(\texttt{la2.x} - \texttt{ls1.x})^2 + (\texttt{la2.y} - \texttt{ls1.y})^2}$$

The latter expression is the actual Mathematical Model associated to the **`ls1`** instance in the Prediction Model. The latter formula depends only on object properties and it can be used at any time to predict the expected **`ls1`** readings. To decide whether or not there is an inconsistency between the model and the reality the calculations results are compared with actual sensor's readings. We suppose that the calculations give 120 lux.

6) The comparison between the actual value (80 lux) or Illuminance reported by the sensor, and the expected value (120 lux) calculated with the prediction model, leads the system to deduce that there is a failure. Either the controlled actuator or the sensor is broken. **This is Fault Detection**. However, we can have another scenario where we have another light sensor in the room. In such a case, if the second light sensor's theoretical value (calculated with its own set of rules from the Prediction Model) is also different from its actual reading, then the faulty component is most probably the light bulb. In the opposite case, if the second light sensor theoretical value is equal (or close enough) to its reading, then the first light sensor is most probably the faulty component. This reasoning to try to find the cause of the fault is **Fault Diagnosis**.

We note here that a more detailed probabilistic approach is introduced in the Perspective section in this dissertation. We reckon that our framework's Prediction Model contains enough information. This information is mainly, the sensor(s) reporting the failure(s) (or symptoms), and the links between actuators and sensors discovered at run-time. The latter allows having a better understanding of the actual dynamic system structure at all time. This information combined allows the performing of Fault Diagnosis using state of the art diagnosis techniques, such as probability.

7) In this scenario we suppose that the system deduces that it is most probably the light bulb that is burnt out. It then sends an error notification to the user. If the latter confirms it, his/her feedback can be added as a statement to the ontology for possible further use. User feedbacks can be used as an amelioration (or correction) for the Fault Diagnosis conclusions. In fact if the user's feedback confirms that the light bulb is properly working despite the fact that the framework reports otherwise, the system deduces that it is in fact the sensor that is not functioning correctly.

6.2. Scenario_1: Light system fault detection and diagnosis

6.2.1. Description of the ambient environment

We start with a square-shaped, 16 square meters(4x4), one-zone room equipped with three light actuators and 2 light sensors positioned as illustrated in **Figure 93**. The x y coordinates have the bottom left corner as an origin.

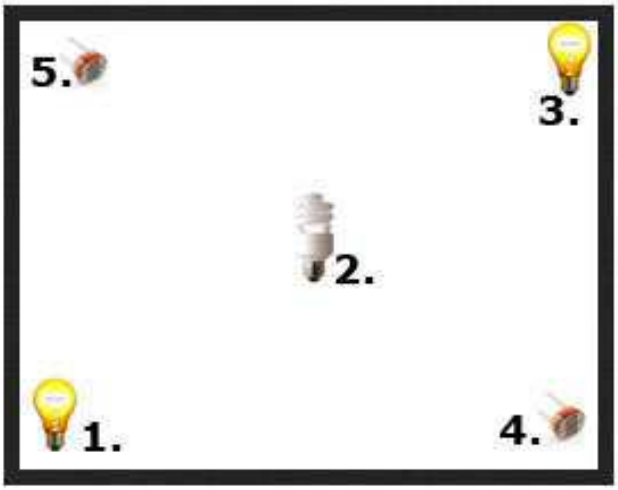

Figure 93. Example 1 of Light system fault detection and diagnosis

In the example depicted in **Figure 93** we have (using **Table 3** to deduce Luminous Flux in *lm* from *Watt* for light bulbs):

- **(1.)** In the bottom left corner, a 40 Watt Incandescent Light Bulb emitting a Luminous Flux Φ of 600 lm; called "Inc Bulb" for short in the rest of the chapter.
- **(2.)** In the center of the room, a 30 Watt Compact Fluorescent Light Bulb emitting a Luminous Flux Φ of 1500 lm; called "CFL Bulb" for short in the rest of the chapter.
- **(3.)** In the top right corner, a 60 Watt Inc Bulb emitting a Luminous Flux Φ of 900 lm.
- **(4.)** In the bottom right corner, a light sensor of type photo diode with a current amplifier with accuracy of ±33%; called "Photo Diode" for short in the rest of the chapter. Note that the exact accuracy is usually defined by the constructor of the component; here we adopt a mean value for Photo Diodes Sensors as described in **[198]**.
- **(5.)** In the top left corner, a Photo Diode Light Sensor with accuracy of ±33%.

Following the finite state machines describing the behaviors of light bulbs defined in the previous chapter, we have the following description of the behavior of the light bulbs.

Inc Bulbs have two possible states: On, in which case they are emitting their defined luminous flux value, and Off, in which case the emitted luminous flux is null.

CFL Bulbs require a slight warm-up time for the electrical current to fully heat the cathodes and reach their full luminous flux output, thus they have four possible states: On and Off that are similar to the description of On and Off of incandescent light bulbs, and the states warming and cooling that are the transitional states between On and Off. The warming state lasts 30 seconds during which the Luminous Flux increases by 50 *lm* each second. The cooling state lasts 60 seconds during which the Luminous Flux is 0. If the CFL bulb is tuned on again before the end of the cooling time it will skip the warming state and go directly to being On.

6.2.2. Building the models

6.2.2.1. The concrete model

In the Concrete Model we define the types and entities that are going to be instantiated later. The instances will be the basis for building the Prediction Model.

To build the concrete model based on the description of the environment given earlier, we identify and separate components that are of type actuators, sensors, and effect modifiers. Then we distinguish between different types of actuators, different types of sensors, and different types of effect modifiers.

The actuators types we identify based on the description of the Ambient Environment are the two types of light bulbs: Inc Bulb and CFL Bulb.

The only light sensor type used in the description is Photo Diode.

There are no effect modifiers in this example.

The Effect that is observed in this example is the Light Effect with one Effect Property Luminous Flux.

The Ambient environment is described in a 2-dimensional space so we would use the 2D Light Law Set and the On/Off Light Law Set.

From this description we can now build the Concrete Model.

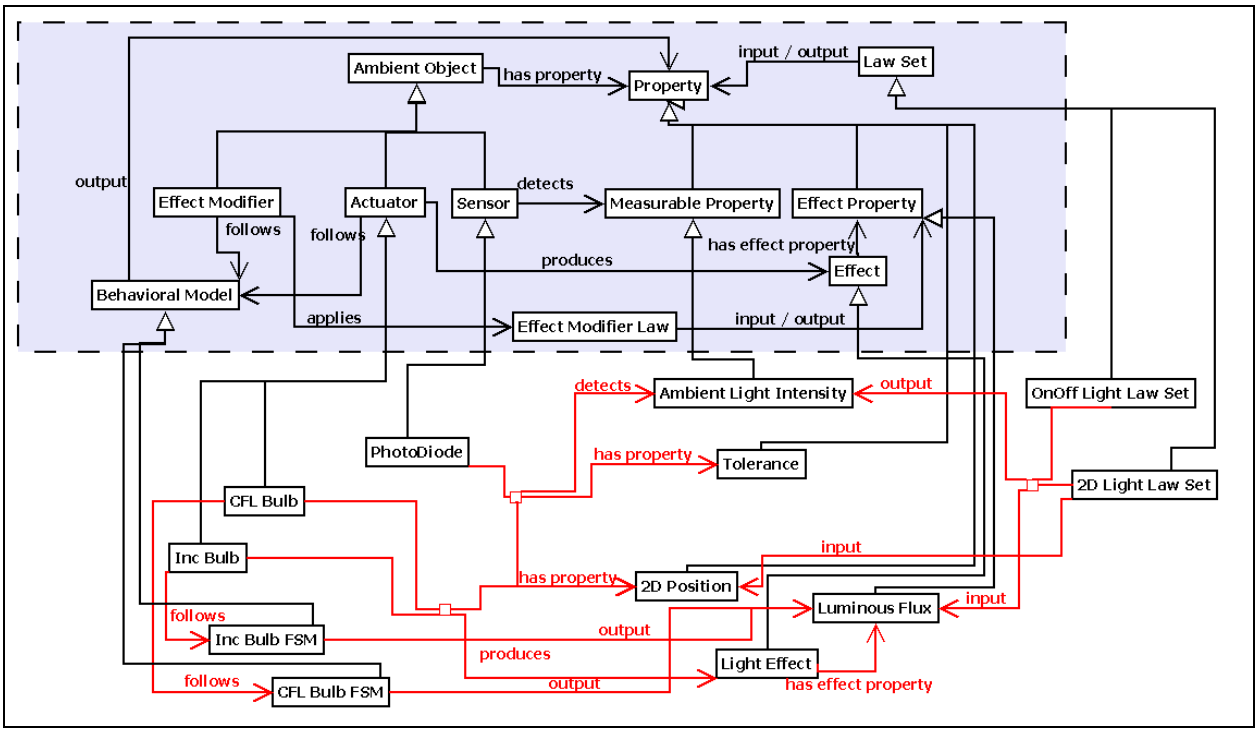

Figure 94. Example 1: Concrete Model for the Ambient Light System

The main entities that are introduced are:

Inc Bulb: an instance of Actuator from the Abstract Model. It represents the Incandescent Light Bulbs type.

CFL Bulb: also is an instantiation of the Abstract Type Actuator representing the Compact Fluorescent Light Bulbs.

Photo Diode: a Sensor type for Photo Diode Light Sensors.

Light Effect: an instantiation of the abstract type Effect. Modeling the physical phenomenon of light waves propagations in the Ambient Environment.

Luminous Flux: instance of the abstract type Effect Property. It is the Light Effect property. It contains the information about the light flux emitted by the components that produce a Light Effect.

Ambient Light Intensity: instance of Measurable Property from the Abstract Model. It contains the value of the reading of the sensor it is connected to. In this case it contains the value of the reading of the Light Sensor. The Prediction Model will try to estimate the value of this entity to perform fault detection.

2D Light Law Set: instance of Law Set containing a set of laws that allows the calculation of the ambient light intensity received by a device according to its 2 dimensional position.

OnOff Light Law Set: another instance of Law Set, to be used when estimating the ambient light intensity value that a device, not having 2 dimensional coordinates, is exposed to.

2D Position: an instance of Property. Actuator types and Sensor types have this property. When an instance does not have a property value for this entity the Prediction Model uses the OnOff Light Law Set with that instance to perform Fault Detection.

Tolerance: also an instance of Property. In this example only Sensor types have this Property.

Inc Bulb FSM: instance of the abstract type Behavioral Model. It is the finite state machine describing the general behavior of Inc Bulbs.

CFL Bulb FSM: also instance of the abstract type Behavioral Model. In the case of CFL Bulbs, it is a timed finite state machine describing the general behavior of CFL Bulbs.

6.2.2.2. The mathematical model

As described earlier, we chose the 2D Light Law Set and the On/Off Light Law Set. However, the environment is also a one-zone environment so in order to reduce the amount of calculations we can consider a sub set of the Law Sets we are going to use where we eliminate the L1 law (verifying objects are in the same zone).

So our final Law Sets are (in order of their hierarchy from most detailed "to try first" to least detailed "to try last"):

2D Light Law Set

$$\text{distance}(s,a) = \begin{cases} \sqrt{(x(s)-x(a))^2 + (y(s)-y(a))^2} & \textit{when } \text{sameZone}(s,a) \textit{ is true} \\ +\infty \quad \textit{when } \text{sameZone}(s,a) \textit{ is false} \end{cases} \tag{L5}$$

$$\text{directLightExposure}(s,a) = \frac{\text{luminousFlux}(a)}{\text{distance}(s,a)^2} \tag{L3}$$

$$\text{ambientLightIntensity}(s) = \sum_a \text{directLightExposure}(s,a) \tag{L4}$$

On Off Light Law Set

$$\text{booleanDirectLightExposure}(s,a) = \begin{cases} true & when\ \text{luminousFlux}(a) \neq 0\ AND\ \text{sameZone}(s,a)\ is\ true \\ false & when\ \text{luminousFlux}(a) \equiv 0\ OR\ \text{sameZone}(s,a)\ is\ false \end{cases} \quad \text{(L6)}$$

$$\text{ambientLightIntensity}(s) = \bigcup_{a \in LightActuators} \text{booleanDirectLightExposure}(s,a) \quad \text{(L7)}$$

6.2.2.3. **The behavioral models**

To describe the behavior of the active and controlled devices in the environment in a formal way we use finite state machines. At instance level, the final values of every instance property depend on the current object state. Note that all objects that are controlled by the Ambient Intelligent System have to be described at least according to the minimal state machine, which is the on-off state machine with two transitions: turn On, turn Off.

The finite state machine for the actuator type Inc Bulb is depicted on **Figure 95**:

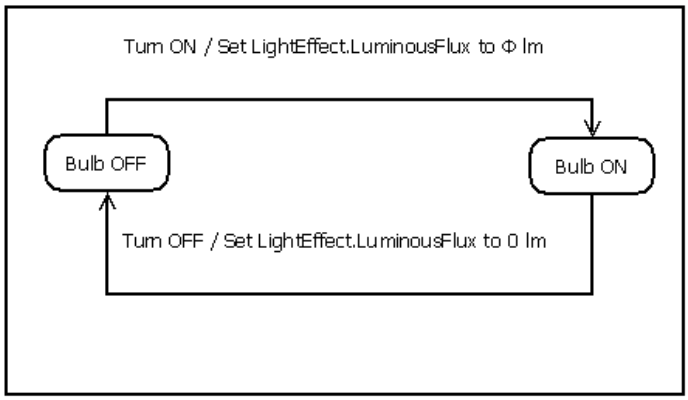

Figure 95. Inc Bulb Finite State Machine

This finite state machine describes the general behavior of Inc Bulbs in general.

When instantiating the Inc Bulbs that are present in the environment their particular Luminous Flux Φ values are defined. Later when the Prediction Model starts evaluating the values of the properties of each Inc Bulb instance using this Finite State Machine, the Luminous Flux Φ is replaced by its value for each instance. So, in the Prediction Model, the Finite state machine for the Inc Bulb (that we will call bulb_1) described by **Figure 93.1.** would look like:

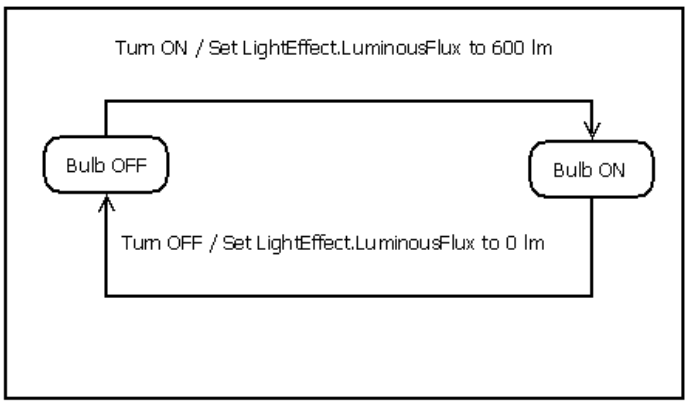

Figure 96. bulb_1 Finite State Machine

The exact value for the Luminous Flux Φ is 600 lm.

The behavior of the compact fluorescent light bulbs, as described in the environment description paragraph, depends on time so we use the timed finite state machine from **Figure 68**.

From the Description given for this CFL Bulb, the values of Φ, α, β, and i are:

Φ: 1500 lm.

α: 30 s.

β: 60 s.

i: 50 lm.

After instantiation of the only CFL Bulb in the environment (**(2.)** in **Figure 93**), the Prediction Model will create the following timed finite state machine for the CFL bulb (that we will call bulb_2) depicted in **Figure 97**:

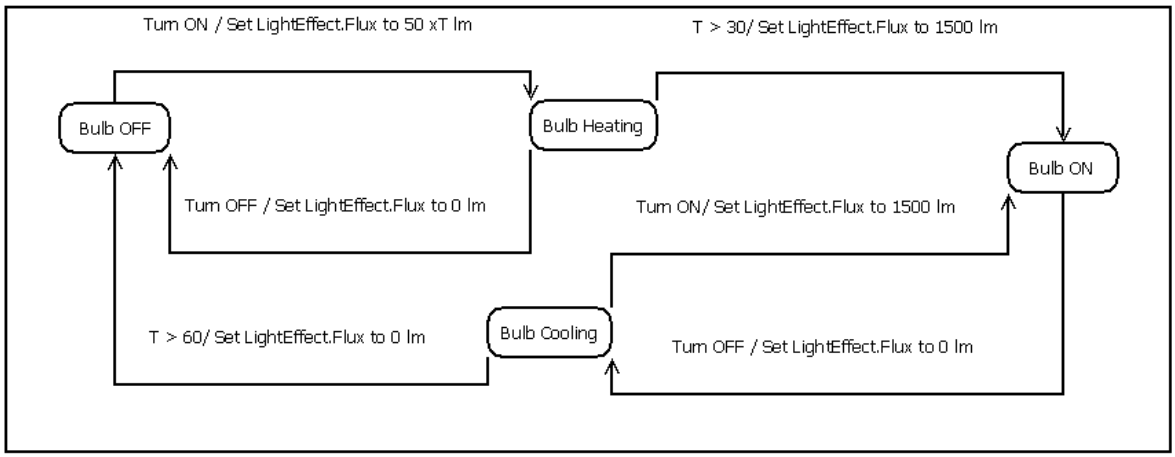

Figure 97. bulb_2 Timed Finite State Machine

As we did for bulb_1, for bulb_2 we replaced the properties of the Bulb Type (here Φ, α, β, and i) with their values for the Bulb Instance.

6.2.3. Instantiating the Models

Based on the description of the Ambient Environment in **Figure 93**, and following the structure of the Concrete Model in **Figure 94**, we create the following instances:

bulb_1: instance of Inc Bulb, with the 2 Dimensional coordinates (0,0), and Φ=600.

bulb_2: instance of CFL Bulb, with the coordinates (2,2), and with Φ=1500.

bulb_3: instance of Inc Bulb, at the position (4,4), and with Φ=900.

lSensor_1: instance of Photo Diode, at the position (4,0), and with tolerance value of 0.33.

lSensor_2: instance of Photo Diode, at the position (0,4), and with tolerance value of 0.33.

6.2.4. Performing fault detection

6.2.4.1. The simulator and building of the Prediction Model

Here we show the triples syntax statements that define the concrete Model and the instances that are used by the simulator to build the Prediction Model.

First the definition of the Concrete Model (depicted in **Figure 94**):

```
FluorescentLightBulb TYPE Actuator;
IncandescentLightBulb TYPE Actuator;

PhotoDiode TYPE Sensor;

AmbientLight TYPE PhysicalProperty;

AmbientLightModificationRatio TYPE PhysicalProperty;

LightFlux TYPE PhysicalProperty;

X TYPE AbsolutePosition;
Y TYPE AbsolutePosition;

LightEffect TYPE Effect;

FluorescentLightBulb produces LightEffect.LightFlux;
IncandescentLightBulb produces LightEffect.LightFlux;

PhotoDiode detects AmbientLight;

FluorescentLightBulb hasProperty X;
FluorescentLightBulb hasProperty Y;

IncandescentLightBulb hasProperty X;
IncandescentLightBulb hasProperty Y;

PhotoDiode hasProperty X;
PhotoDiode hasProperty Y;

PhotoDiode hasTolerance 33;

AmbientLightLawSet2D TYPE LawSet;

   #ambientLightIntensity function is called here TotalIllumination2D
   #to be compatible with corresponding java function.
   #Different java function were defined for 3D, 2D, and onOff
   #similar redefining was done for the other functions of the Law-Sets

TotalIllumination2D TYPE Function;
TotalIllumination2D hasOutput AmbientLight;
TotalIllumination2D hasExpression _SUM(a, LightActuator, SingleIllumination2D(a,
   sensor));
TotalIllumination2D hasArg sensor;

AmbientLightLawSet2D hasFunction TotalIllumination2D;

SingleIllumination2D TYPE Function;
SingleIllumination2D hasOutput singleSourceLight;
SingleIllumination2D hasExpression (actuator.LightFlux)/(Distance2D(actuator,
   sensor)^2);
SingleIllumination2D hasArg actuator;
SingleIllumination2D hasArg sensor;

AmbientLightLawSet2D hasFunction SingleIllumination2D;

Distance2D TYPE Function;
Distance2D hasOutput distance;
Distance2D hasExpression (((a.X-b.X)^2)+((a.Y-b.Y)^2))^(0.5);
Distance2D hasArg a;
Distance2D hasArg b;

AmbientLightLawSet2D hasFunction Distance2D;

LightEffect hasLawSet AmbientLightLawSet2D;
```

```
AmbientLightLawSetOnOff TYPE LawSet;

TotalIlluminationOnOff TYPE Function;
TotalIlluminationOnOff hasOutput AmbientLight;
TotalIlluminationOnOff hasExpression _OR(a, LightActuator,
   SingleIlluminationOnOff(a, sensor));
TotalIlluminationOnOff hasArg sensor;

AmbientLightLawSetOnOff hasFunction TotalIlluminationOnOff;

SingleIlluminationOnOff TYPE Function;
SingleIlluminationOnOff hasOutput SingleSourceLight;
SingleIlluminationOnOff hasExpression actuator.LightFlux;
SingleIlluminationOnOff hasArg actuator;
SingleIlluminationOnOff hasArg sensor;

AmbientLightLawSetOnOff hasFunction SingleIlluminationOnOff;

LightEffect hasLawSet AmbientLightLawSetOnOff;

FluorescentLightBulb FSM resources/FSM/FluorescentLightBulb_FSM.xml;
IncandescentLightBulb FSM resources/FSM/IncandescentLightBulb_FSM.xml;
```

Then the declaration of the instances:

```
bulb_1 is IncandescentLightBulb;
bulb_1 X 0;
bulb_1 Y 0;
bulb_1 LightFlux 600;
bulb_2 is FluorescentLightBulb;
bulb_2 X 2;
bulb_2 Y 2;
bulb_2 LightFlux 1500;
bulb_3 is IncandescentLightBulb;
bulb_3 X 4;
bulb_3 Y 4;
bulb_3 LightFlux 900;
lSensor_1 is PhotoDiode;
lSensor_1 X 4;
lSensor_1 Y 0;
lSensor_1 AmbientLight 20;
lSensor_2 is PhotoDiode;
lSensor_2 X 0;
lSensor_2 Y 4;
lSensor_2 AmbientLight 25;
```

Note the complete decoupling between Actuators and Sensors in the previous definition of the Concrete Model and of the Instances. The links are going to be deduces automatically once the simulation is running, due to the evaluation of the Mathematical Model. The links are in fact different Properties from Actuators and Sensors that are exploited in the Mathematical Functions of the Law-Sets.

6.2.4.2. **The simulation**

In order to facilitate our series of tests we use a simulation instead of real devices. The real devices behavior is simulated by our simulation application. The module simulating the behavior of the devices is decoupled from the core of the FDD framework and thus can be easily replaced by an interface for connecting real devices (hardware layer) with the FDD framework.

The goal of this simulation is to observe the running of the fault detection task of the FDD framework using ModHel'X in order to validate the correctness of our theoretical study.

Using the previous description of the Concrete Model and the instances the simulator creates a Prediction Model that is converted to ModHel'X forma in order to run the simulation while visualizing the Prediction Model and the time units' steps.

Figure 98 depicts how the Prediction Model of **Scenario 1** looks like in ModHel'X:

Figure 99 is the control panel for controlling the values of the devices instances parameters, and for triggering state transitions in the instantiated finite state machines of the actuators.

Figure 100 is the trace output of the Prediction Model, it depicts the sensors predicted readings estimated via the Calculation Block that contains the instantiated Mathematical Model..

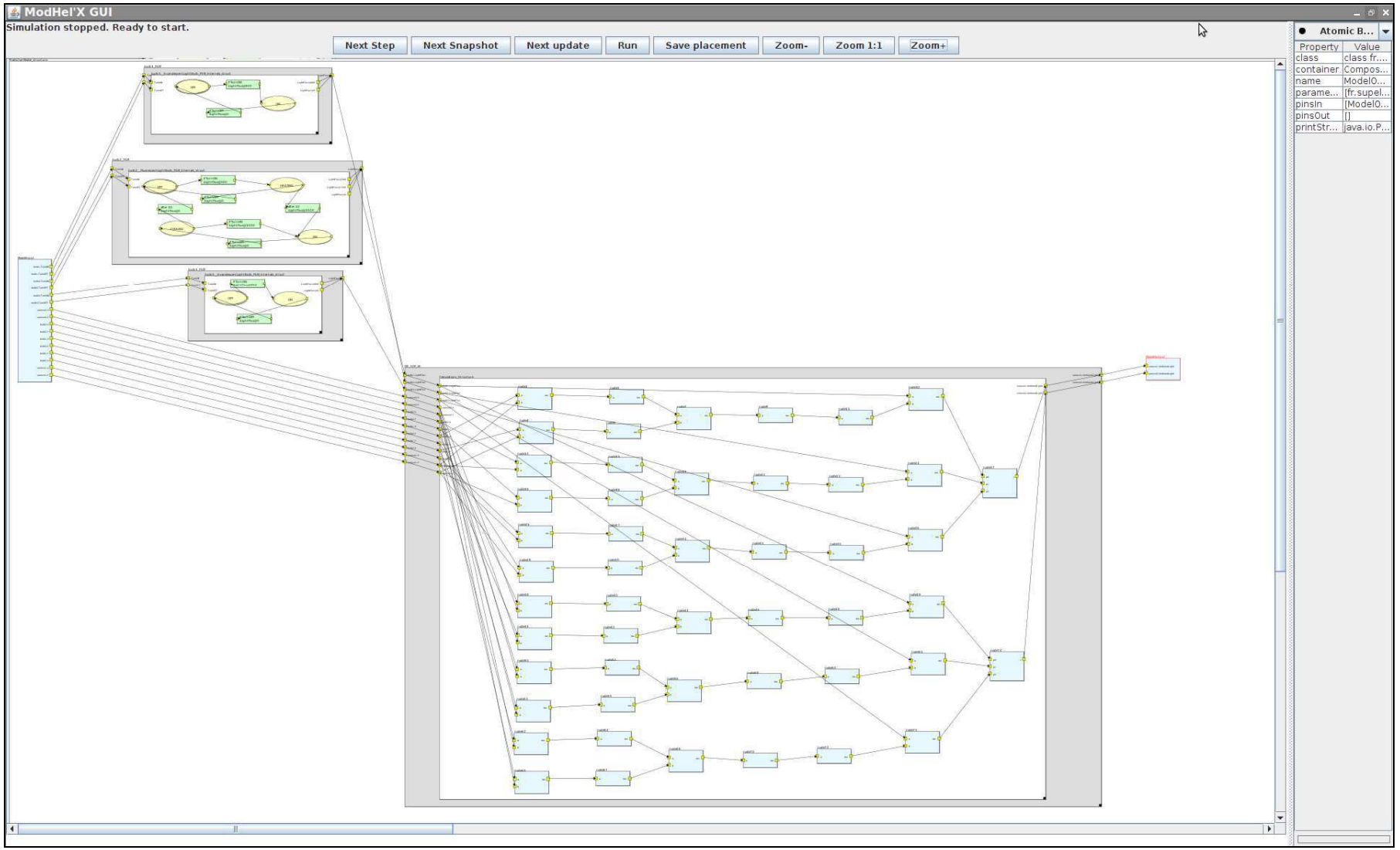

Figure 98. General View of the ModHel'X representation of the Prediction Model for Scenario_1

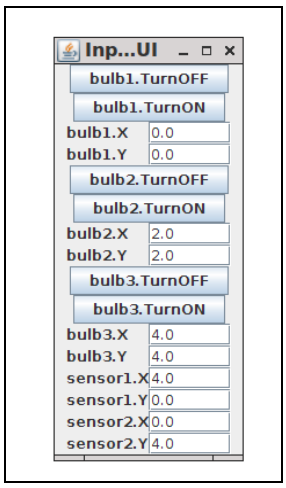

Figure 99. Control panel for the Prediction Model inputs – scenario_1

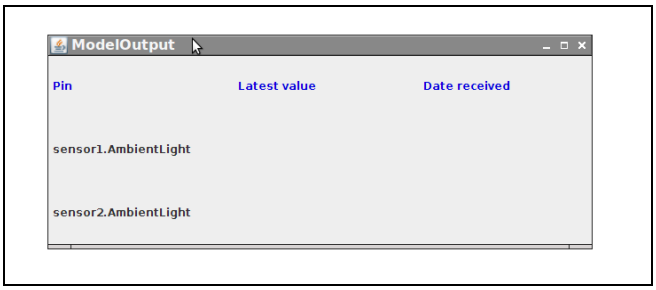

Figure 100. Prediction Model's output trace window – scenario_1

- <u>Validating the Model:</u>

Next we run the simulation, during which we perform a series of tests to verify the correctness of the Prediction Model's calculations.

<u>Test 1</u>: We start with the light bulbs in their initial state Off, which means the luminous flux Φ for both light bulbs equals 0 lm. We observe the following output:

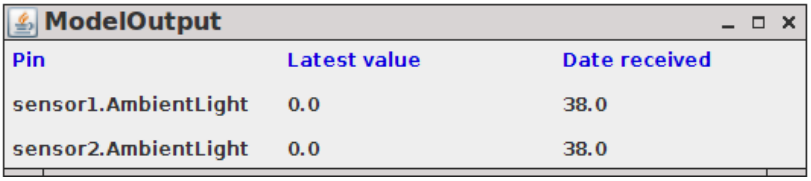

Figure 101. Scenario_1 Test1 simulation trace values

This verifies the manual calculations for the predicted reading for the sensors; since the law **L3**, applied to sensor_1 or to sensor_2, is always equal to 0 for both light bulbs, which means that the sum calculated by **L4** would also be equal to 0 for both sensors.

Test 2: We turn on bulb_1 (emitting light flux Φ = 600 lm) and we observe a change in sensors' predicted readings:

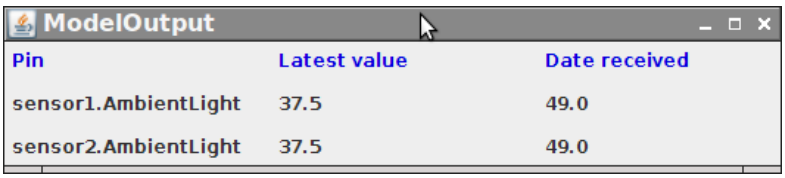

Figure 102. Scenario_1 Test2 simulation trace values

Having the same predicted reading value for both sensors is verifiable since bulb_1 is at equal distance from both sensors.

Evaluating the Mathematical Model and Validation: The value calculated by **L4** for sensor_1 should be 37.5. The 2D Light Law Set calls are as following:

Call to L4:

```
ambientLightIntensity(sensor_1)=
directLightExposure(sensor_1,bulb_1)+
directLightExposure(sensor_1,bulb_2)+
directLightExposure(sensor_1,bulb_3)
```

- this generates 3 calls to the Law L3; one for each bulb

First call to L3:

```
directLightExposure(sensor_1,bulb_1)=
luminousFlux(bulb_1) / distance(sensor_1,bulb_1)^2
```

Second call to L3:

```
directLightExposure(sensor_1,bulb_2)=
luminousFlux(bulb_2) / distance(sensor_1,bulb_2)^2
```

Third call to L3:

```
directLightExposure(sensor_1,bulb_3)=
luminousFlux(bulb_3) / distance(sensor_1,bulb_3)^2
```

with

```
luminousFlux(bulb_1)=600
luminousFlux(bulb_2)=0
luminousFlux(bulb_3)=0
```

- this generates 3 calls to the Law **L5**; one for each bulb

First call to L5:

```
distance(sensor_1,bulb_1)=
Sqrt[(x(sensor_1)-x(bulb_1))^2 + (y(sensor_1)-y(bulb_1))^2]
```

Second call to L5:

```
distance(sensor_1,bulb_2)=
Sqrt[(x(sensor_1)-x(bulb_2))^2 + (y(sensor_1)-y(bulb_2))^2]
```

Third call to L5:

```
distance(sensor_1,bulb_3)=
Sqrt[(x(sensor_1)-x(bulb_3))^2 + (y(sensor_1)-y(bulb_3))^2]
```

with

x(sensor_1)=4 y(sensor_1)=0
x(bulb_1)=0 y(bulb_1)=0
x(bulb_2)=2 y(bulb_2)=2
x(bulb_3)=4 y(bulb_3)=4

which give

distance(sensor_1,bulb_1)=4
distance(sensor_1,bulb_2)=2.82
distance(sensor_1,bulb_3)=4

using these values to evaluate previous L3 calls gives

directLightExposure(sensor_1,bulb_1)=600/16
directLightExposure(sensor_1,bulb_2)=0/8
directLightExposure(sensor_1,bulb_3)=0/16

Finally the call to L4 for sensor_1 gives

ambientLightIntensity(sensor_1)=37.5+0+0= 37.5

That is equal to the value predicted by the Prediction Model.

As a possible optimization to the simulation engine calculations we can choose to ignore the second and third call to L5 as the luminous flux value for bulb_2 and bulb_3 equals 0. This could alleviate a lot of calculation charges when we have too many devices to consider. We would ignore those who generate 0 effects in the environment.

Test 3: Then we turn on bulb_3 along side with bulb_1. We obtain the following output:

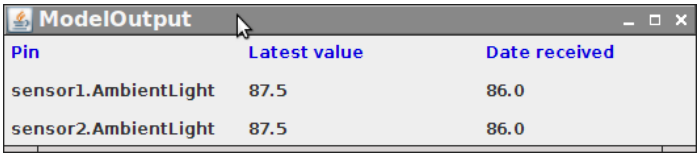

Figure 103. Scenario_1 Test3 simulation trace values

The distance between sensor_1 and bulb_1 is equal to the distance between sensor_2 and bulb_1, and the distance between sensor_1 and bulb_3 is equal to the distance between sensor_2 and bulb_3, which justifies having the same predicted value for both sensors. We verify the value of the predicted sensor_1 reading.

Evaluating the Mathematical Model and Validation: The value calculated by **L4** for sensor_1 should be 87.5. The 2D Light Law Set calls are as following:

Call to L4:

ambientLightIntensity(sensor_1)=
directLightExposure(sensor_1,bulb_1)+
directLightExposure(sensor_1,bulb_2)+
directLightExposure(sensor_1,bulb_3)

- this generates 3 calls to the Law L3; one for each bulb

First call to L3:

directLightExposure(sensor_1,bulb_1)=

luminousFlux(bulb_1) / distance(sensor_1,bulb_1)^2

Second call to L3:

directLightExposure(sensor_1,bulb_2)=
luminousFlux(bulb_2) / distance(sensor_1,bulb_2)^2

Third call to L3:

directLightExposure(sensor_1,bulb_3)=
luminousFlux(bulb_3) / distance(sensor_1,bulb_3)^2

with

luminousFlux(bulb_1)=600
luminousFlux(bulb_2)=0
luminousFlux(bulb_3)=800

- this generates 3 calls to the Law L5; one for each bulb

First call to L5:

distance(sensor_1,bulb_1)=
Sqrt[(x(sensor_1)-x(bulb_1))^2 + (y(sensor_1)-y(bulb_1))^2]

Second call to L5:

distance(sensor_1,bulb_2)=
Sqrt[(x(sensor_1)-x(bulb_2))^2 + (y(sensor_1)-y(bulb_2))^2]

Third call to L5:

distance(sensor_1,bulb_3)=
Sqrt[(x(sensor_1)-x(bulb_3))^2 + (y(sensor_1)-y(bulb_3))^2]

with

x(sensor_1)=4 y(sensor_1)=0
x(bulb_1)=0 y(bulb_1)=0
x(bulb_2)=2 y(bulb_2)=2
x(bulb_3)=4 y(bulb_3)=4

which give

distance(sensor_1,bulb_1)=4
distance(sensor_1,bulb_2)=2.82
distance(sensor_1,bulb_3)=4

using these values to evaluate previous L3 calls gives

directLightExposure(sensor_1,bulb_1)=600/16
directLightExposure(sensor_1,bulb_2)=0/8
directLightExposure(sensor_1,bulb_3)=800/16

Finally the call to L4 for sensor_1 gives

ambientLightIntensity(sensor_1)=37.5+0+50= 87.5

That is equal to the value predicted by the Prediction Model.

Test 4: We move bulb_1 2 meters in the positive direction of the x axis. This generates the following output:

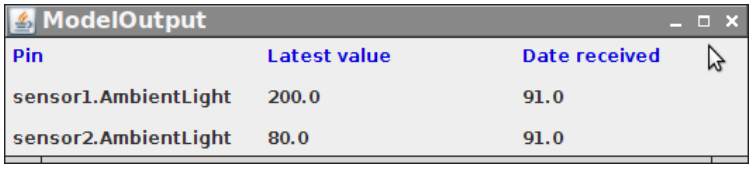

Figure 104. Scenario_1 Test4 simulation trace values

Evaluating the Mathematical Model and Validation: The value calculated by **L4** for sensor_1 should be 200. The 2D Light Law Set calls are as follows:

Call to L4:

```
ambientLightIntensity(sensor_1)=
directLightExposure(sensor_1,bulb_1)+
directLightExposure(sensor_1,bulb_2)+
directLightExposure(sensor_1,bulb_3)
```

- this generates 3 calls to the Law L3; one for each bulb

First call to L3:

```
directLightExposure(sensor_1,bulb_1)=
luminousFlux(bulb_1) / distance(sensor_1,bulb_1)^2
```

Second call to L3:

```
directLightExposure(sensor_1,bulb_2)=
luminousFlux(bulb_2) / distance(sensor_1,bulb_2)^2
```

Third call to L3:

```
directLightExposure(sensor_1,bulb_3)=
luminousFlux(bulb_3) / distance(sensor_1,bulb_3)^2
```

with

```
luminousFlux(bulb_1)=600
 luminousFlux(bulb_2)=0
luminousFlux(bulb_3)=800
```

- this generates 3 calls to the Law L5: one for each bulb.

First call to L5:

```
distance(sensor_1,bulb_1)=
Sqrt[(x(sensor_1)-x(bulb_1))^2 + (y(sensor_1)-y(bulb_1))^2]
```

Second call to L5:

```
distance(sensor_1,bulb_2)=
Sqrt[(x(sensor_1)-x(bulb_2))^2 + (y(sensor_1)-y(bulb_2))^2]
```

Third call to L5:

```
distance(sensor_1,bulb_3)=
Sqrt[(x(sensor_1)-x(bulb_3))^2 + (y(sensor_1)-y(bulb_3))^2]
```

with

```
x(sensor_1)=4 y(sensor_1)=0
  x(bulb_1)=2 y(bulb_1)=0
  x(bulb_2)=2 y(bulb_2)=2
  x(bulb_3)=4 y(bulb_3)=4
```

which gives

```
distance(sensor_1,bulb_1)=2
```

distance(sensor_1,bulb_2)=2.82

distance(sensor_1,bulb_3)=4

using these values to evaluate previous L3 calls gives

directLightExposure(sensor_1,bulb_1)=600/4

directLightExposure(sensor_1,bulb_2)=0/8

directLightExposure(sensor_1,bulb_3)=800/16

Finally the call to L4 for sensor_1 gives

ambientLightIntensity(sensor_1)=150+0+50= 200

That is equal to the value predicted by the Prediction Model.

The bulbs are no longer at equal distance from the sensors, so we calculate the value calculated by **L4** for sensor_2 also, which should be 80. The 2D Light Law Set calls are as following:

Call to L4:

ambientLightIntensity(sensor_2)=
directLightExposure(sensor_2,bulb_1)+
directLightExposure(sensor_2,bulb_2)+
directLightExposure(sensor_2,bulb_3)

- this generates 3 calls to the Law L3; one for each bulb

First call to L3:

directLightExposure(sensor_2,bulb_1)=
luminousFlux(bulb_1) / distance(sensor_2,bulb_1)^2

Second call to L3:

directLightExposure(sensor_1,bulb_2)=
luminousFlux(bulb_2) / distance(sensor_2,bulb_2)^2

Third call to L3:

directLightExposure(sensor_1,bulb_3)=
luminousFlux(bulb_3) / distance(sensor_2,bulb_3)^2

with

luminousFlux(bulb_1)=600

luminousFlux(bulb_2)=0

luminousFlux(bulb_3)=800

- this generates 3 calls to the Law L5; one for each bulb

First call to L5:

distance(sensor_2,bulb_1)=
Sqrt[(x(sensor_2)-x(bulb_1))^2 + (y(sensor_2)-y(bulb_1))^2]

Second call to L5:

distance(sensor_1,bulb_2)=
Sqrt[(x(sensor_2)-x(bulb_2))^2 + (y(sensor_2)-y(bulb_2))^2]

Third call to L5:

distance(sensor_2,bulb_3)=
Sqrt[(x(sensor_2)-x(bulb_3))^2 + (y(sensor_2)-y(bulb_3))^2]

with

x(sensor_2)=0 y(sensor_2)=4

x(bulb_1)=2 y(bulb_1)=0
x(bulb_2)=2 y(bulb_2)=2
x(bulb_3)=4 y(bulb_3)=4

which give

distance(sensor_2,bulb_1)=4.47
distance(sensor_2,bulb_2)=2.82
distance(sensor_2,bulb_3)=4

using these values to evaluate previous L3 calls gives

directLightExposure(sensor_2,bulb_1)=600/20
directLightExposure(sensor_2,bulb_2)=0/8
directLightExposure(sensor_2,bulb_3)=800/16

Finally the call to L4 for sensor_1 gives

ambientLightIntensity(sensor_2)=30+0+50= 80

That is equal to the value predicted by the Prediction Model.

Test 5: We turn on bulb_2, the first second we should observe a 50 lm effect on the sensors. The consequence of this effect is the following:

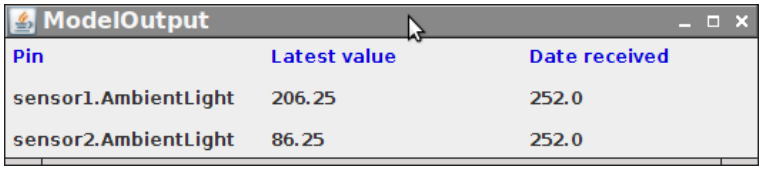

Figure 105. Scenario_1 Test5 simulation trace values

Evaluating the Mathematical Model and Validation: The value calculated by **L4** for sensor_1 should be 206.25. The 2D Light Law Set calls are as following:

Call to L4:

ambientLightIntensity(sensor_1)=
directLightExposure(sensor_1,bulb_1)+
directLightExposure(sensor_1,bulb_2)+
directLightExposure(sensor_1,bulb_3)

- this generates 3 calls to the Law L3; one for each bulb

First call to L3:

directLightExposure(sensor_1,bulb_1)=
luminousFlux(bulb_1) / distance(sensor_1,bulb_1)^2

Second call to L3:

directLightExposure(sensor_1,bulb_2)=
luminousFlux(bulb_2) / distance(sensor_1,bulb_2)^2

Third call to L3:

directLightExposure(sensor_1,bulb_3)=
luminousFlux(bulb_3) / distance(sensor_1,bulb_3)^2

with

luminousFlux(bulb_1)=600
luminousFlux(bulb_2)=50

luminousFlux(bulb_3)=800

- this generates 3 calls to the Law L5; one for each bulb

First call to L5:

distance(sensor_1,bulb_1)=
Sqrt[(x(sensor_1)-x(bulb_1))^2 + (y(sensor_1)-y(bulb_1))^2]

Second call to L5:

distance(sensor_1,bulb_2)=
Sqrt[(x(sensor_1)-x(bulb_2))^2 + (y(sensor_1)-y(bulb_2))^2]

Third call to L5:

distance(sensor_1,bulb_3)=
Sqrt[(x(sensor_1)-x(bulb_3))^2 + (y(sensor_1)-y(bulb_3))^2]

with

x(sensor_1)=4 y(sensor_1)=0
x(bulb_1)=2 y(bulb_1)=0
x(bulb_2)=2 y(bulb_2)=2
x(bulb_3)=4 y(bulb_3)=4

which give

distance(sensor_1,bulb_1)=2
distance(sensor_1,bulb_2)=2.82
distance(sensor_1,bulb_3)=4

using these values to evaluate previous L3 calls gives

directLightExposure(sensor_1,bulb_1)=600/4
directLightExposure(sensor_1,bulb_2)=50/8
directLightExposure(sensor_1,bulb_3)=800/16

Finally the call to L4 for sensor_1 gives

ambientLightIntensity(sensor_1)=150+6.25+50= 206.25

That is equal to the value predicted by the Prediction Model.

The bulbs are no longer at equal distance from the sensors, so we calculate the value calculated by **L4** for sensor_2 also, which should be 86.25. The 2D Light Law Set calls are as following:

Call to L4:

ambientLightIntensity(sensor_2)=
directLightExposure(sensor_2,bulb_1)+
directLightExposure(sensor_2,bulb_2)+
directLightExposure(sensor_2,bulb_3)

- this generates 3 calls to the Law L3; one for each bulb

First call to L3:

directLightExposure(sensor_2,bulb_1)=
luminousFlux(bulb_1) / distance(sensor_2,bulb_1)^2

Second call to L3:

directLightExposure(sensor_1,bulb_2)=
luminousFlux(bulb_2) / distance(sensor_2,bulb_2)^2

Third call to L3:

directLightExposure(sensor_1,bulb_3)=
luminousFlux(bulb_3) / distance(sensor_2,bulb_3)^2

with

luminousFlux(bulb_1)=600
luminousFlux(bulb_2)=50
luminousFlux(bulb_3)=800

- this generates 3 calls to the Law L5; one for each bulb

First call to L5:

distance(sensor_2,bulb_1)=
Sqrt[(x(sensor_2)-x(bulb_1))^2 + (y(sensor_2)-y(bulb_1))^2]

Second call to L5:

distance(sensor_1,bulb_2)=
Sqrt[(x(sensor_2)-x(bulb_2))^2 + (y(sensor_2)-y(bulb_2))^2]

Third call to L5:

distance(sensor_2,bulb_3)=
Sqrt[(x(sensor_2)-x(bulb_3))^2 + (y(sensor_2)-y(bulb_3))^2]

with

x(sensor_2)=0 y(sensor_2)=4
x(bulb_1)=2 y(bulb_1)=0
x(bulb_2)=2 y(bulb_2)=2
x(bulb_3)=4 y(bulb_3)=4

which give

distance(sensor_2,bulb_1)=4.47
distance(sensor_2,bulb_2)=2.82
distance(sensor_2,bulb_3)=4

using these values to evaluate previous L3 calls gives

directLightExposure(sensor_2,bulb_1)=600/20
directLightExposure(sensor_2,bulb_2)=50/8
directLightExposure(sensor_2,bulb_3)=800/16

Finally the call to L4 for sensor_1 gives

ambientLightIntensity(sensor_2)=30+6.25+50= 86.25

That is equal to the value predicted by the Prediction Model.

Test 6: After 30 seconds of turning on bulb_2 we should observe the 1500 lm effect on the sensors, in fact we observe:

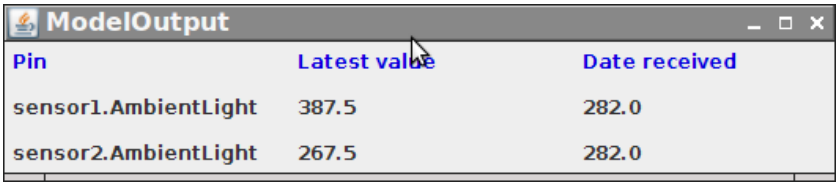

Figure 106. Scenario_1 Test6 simulation trace values

Evaluating the Mathematical Model and Validation: The value calculated by **L4** for sensor_1 should be 287.5. The 2D Light Law Set calls are as following:

Call to L4:

```
ambientLightIntensity(sensor_1)=
directLightExposure(sensor_1,bulb_1)+
directLightExposure(sensor_1,bulb_2)+
directLightExposure(sensor_1,bulb_3)
```

- this generates 3 calls to the Law L3; one for each bulb

First call to L3:

```
directLightExposure(sensor_1,bulb_1)=
luminousFlux(bulb_1) / distance(sensor_1,bulb_1)^2
```

Second call to L3:

```
directLightExposure(sensor_1,bulb_2)=
luminousFlux(bulb_2) / distance(sensor_1,bulb_2)^2
```

Third call to L3:

```
directLightExposure(sensor_1,bulb_3)=
luminousFlux(bulb_3) / distance(sensor_1,bulb_3)^2
```

with

```
luminousFlux(bulb_1)=600
luminousFlux(bulb_2)=1500
luminousFlux(bulb_3)=800
```

- this generates 3 calls to the Law L5; one for each bulb

First call to L5:

```
distance(sensor_1,bulb_1)=
Sqrt[(x(sensor_1)-x(bulb_1))^2 + (y(sensor_1)-y(bulb_1))^2]
```

Second call to L5:

```
distance(sensor_1,bulb_2)=
Sqrt[(x(sensor_1)-x(bulb_2))^2 + (y(sensor_1)-y(bulb_2))^2]
```

Third call to L5:

```
distance(sensor_1,bulb_3)=
Sqrt[(x(sensor_1)-x(bulb_3))^2 + (y(sensor_1)-y(bulb_3))^2]
```

with

```
x(sensor_1)=4 y(sensor_1)=0
x(bulb_1)=2 y(bulb_1)=0
x(bulb_2)=2 y(bulb_2)=2
x(bulb_3)=4 y(bulb_3)=4
```

which give

```
distance(sensor_1,bulb_1)=2
distance(sensor_1,bulb_2)=2.82
distance(sensor_1,bulb_3)=4
```

using these values to evaluate previous L3 calls gives

```
directLightExposure(sensor_1,bulb_1)=600/4
directLightExposure(sensor_1,bulb_2)=1500/8
directLightExposure(sensor_1,bulb_3)=800/16
```

Finally the call to L4 for sensor_1 gives

ambientLightIntensity(sensor_1)=150+187.5+50= 387.5

That is equal to the value predicted by the Prediction Model.

The bulbs are no longer at equal distance from the sensors, so we calculate the value calculated by **L4** for sensor_2 also, which should be 267.5. The 2D Light Law Set calls are as following:

Call to L4:

ambientLightIntensity(sensor_2)=
directLightExposure(sensor_2,bulb_1)+
directLightExposure(sensor_2,bulb_2)+
directLightExposure(sensor_2,bulb_3)

- this generates 3 calls to the Law L3; one for each bulb

First call to L3:

directLightExposure(sensor_2,bulb_1)=
luminousFlux(bulb_1) / distance(sensor_2,bulb_1)^2

Second call to L3:

directLightExposure(sensor_1,bulb_2)=
luminousFlux(bulb_2) / distance(sensor_2,bulb_2)^2

Third call to L3:

directLightExposure(sensor_1,bulb_3)=
luminousFlux(bulb_3) / distance(sensor_2,bulb_3)^2

with

luminousFlux(bulb_1)=600
luminousFlux(bulb_2)=1500
luminousFlux(bulb_3)=800

- this generates 3 calls to the Law L5; one for each bulb

First call to L5:

distance(sensor_2,bulb_1)=
Sqrt[(x(sensor_2)-x(bulb_1))^2 + (y(sensor_2)-y(bulb_1))^2]

Second call to L5:

distance(sensor_1,bulb_2)=
Sqrt[(x(sensor_2)-x(bulb_2))^2 + (y(sensor_2)-y(bulb_2))^2]

Third call to L5:

distance(sensor_2,bulb_3)=
Sqrt[(x(sensor_2)-x(bulb_3))^2 + (y(sensor_2)-y(bulb_3))^2]

with

x(sensor_2)=0 y(sensor_2)=4
x(bulb_1)=2 y(bulb_1)=0
x(bulb_2)=2 y(bulb_2)=2
x(bulb_3)=4 y(bulb_3)=4

which give

distance(sensor_2,bulb_1)=4.47
distance(sensor_2,bulb_2)=2.82
distance(sensor_2,bulb_3)=4

using these values to evaluate previous L3 calls gives

```
directLightExposure(sensor_2,bulb_1)=600/20
directLightExposure(sensor_2,bulb_2)=1500/8
directLightExposure(sensor_2,bulb_2)=800/16
```

Finally the call to L4 for sensor_1 gives

```
ambientLightIntensity(sensor_2)=30+0+50= 267.5
```

That is equal to the value predicted by the Prediction Model.

Test 7: We turn off bulb_2 we should observe the same readings as in **Figure 104** of **Test 4**.

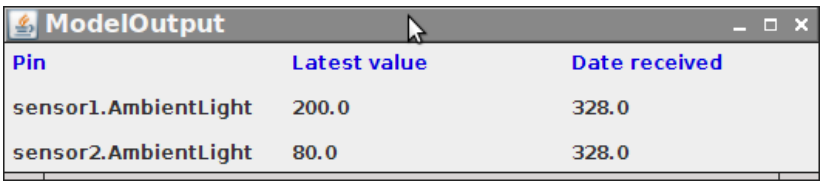

Figure 107. Scenario_1 Test7 simulation trace values - (1/2)

Those readings are verified in **Figure 107**.

After turning off bulb_2, and before the passing of the 60 seconds cooling time described by the bulb_3 finite state machine we turn it on again, we should observe the same readings as Test 6, without passing by the heating phase as in Test 5. This is also verified as we obtain the following predicted readings (same as in Test 5):

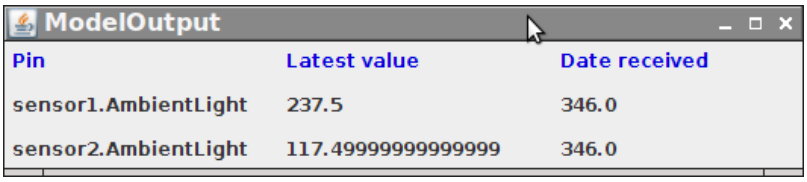

Figure 108. Scenario_1 Test7 simulation trace values - (2/2)

We can conclude that the Prediction Model mathematical predictions and finite state machine described behaviors verify the theoretical calculations for this scenario. In the next scenario we generate a Prediction Model that takes into account the concept of Effect Modifier when predicting sensors' readings.

6.3. Scenario_2: Light system fault detection and diagnosis with Effect Modifier

6.3.1. Description of the smart environment

We consider the environment illustrated in **Figure 109**, in which we have two rooms (room 1 and room 2) separated by a wall.

In **room 1** we have (as described on **Figure 109**):

- **(1.)** a 120 Watt incandescent light bulb with luminous flux Φ of 1750 lm.

In **room 2** we have (also as described on **Figure 109**):

- **(2.)** a 23 Watt compact fluorescent light bulb emitting a luminous flux of 1500 lm.
- **(3.)** a 60 Watt incandescent light bulb emitting 800 lm.

Incandescent light bulbs have two possible states: On, in which case they are emitting their defined luminous flux value, and Off, in which case the emitted luminous flux is null.

Compact fluorescent light bulbs require a slight warm-up time for the electrical current to fully heat the cathodes and reach their full luminous flux output, thus they have four possible states: On and Off that are similar to the description of On and Off of incandescent light bulbs, and the states warming and cooling that are the transitional states between On and Off.

In both rooms we have ((as depicted on **Figure 109**) **(4.)** and **(5.)** photo diodes with a current amplifier light sensor (with accuracy of ±33% **[198]**; called "Photo Diode" for short in the rest of the example).

In addition we have **(6.)** a photo transistor (with accuracy of ±75%) reporting the light intensity outside the two rooms.

room 1 and **room 2** are connected with **(7.)** a double hinged door that opens into room 2. The double hinged door can have 4 possible states: Closed, open, partially open on the right and partially open on the left.

room 1 has a window that opens on the outside. The window is equipped with **(8.)** electric window blinds. The state of the window blinds is independent from that of the window and they can be either up (open) or down (shut).

room 1 also has **(9.)** an electric sliding door that opens to the outside. The door can be either fully open or fully closed.

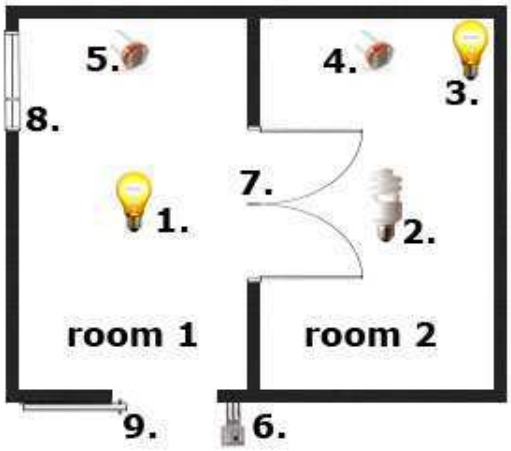

Figure 109. Ambient environment light example

6.3.2. Building the models

6.3.2.1. The concrete model

In this part we build the environment models. The models will be the basis for creating the proper instances representing the components described in **Figure 109**. The fault detection and diagnosis will be performed on those instances.

The first step in identifying the entities in the Concrete Model would be to separate actuators and sensors, then to distinguish between different types of actuators and different types of sensors. We suppose that actuators and sensors are objects that are necessarily controlled by the system. In general, sensors send the system readings reporting the state of the environment, and actuators are objects that act upon the environment via their actions. In this light system fault detection and diagnosis example all components (from **(1.)** to **(9.)**) are considered with respect to light. In a first phase we distinguish the types: light actuators (**(1.)**, **(2.)** and **(3.)**), light sensors (**(4.)**, **(5.)** and **(6.)**) and light bridges (**(7.)**, **(8.)** and **(9.)**).

From the characteristic and behavioral description of the components in the previous paragraph we can furthermore refine the types of light actuators, light sensors and light bridge.

As illustrated in the concrete model on **Figure 110**, we distinguish two types of light actuators: incandescent light bulbs (**(1.)** and **(3.)**), and compact fluorescent light bulbs **(2.)**; two types of light sensors: photo diodes (**(4.)** and **(5.)**), and photo transistors **(6.)**; and three types of light bridges: double hinged doors **(7.)**, window blinds **(8.)**, and sliding doors **(9.)**.

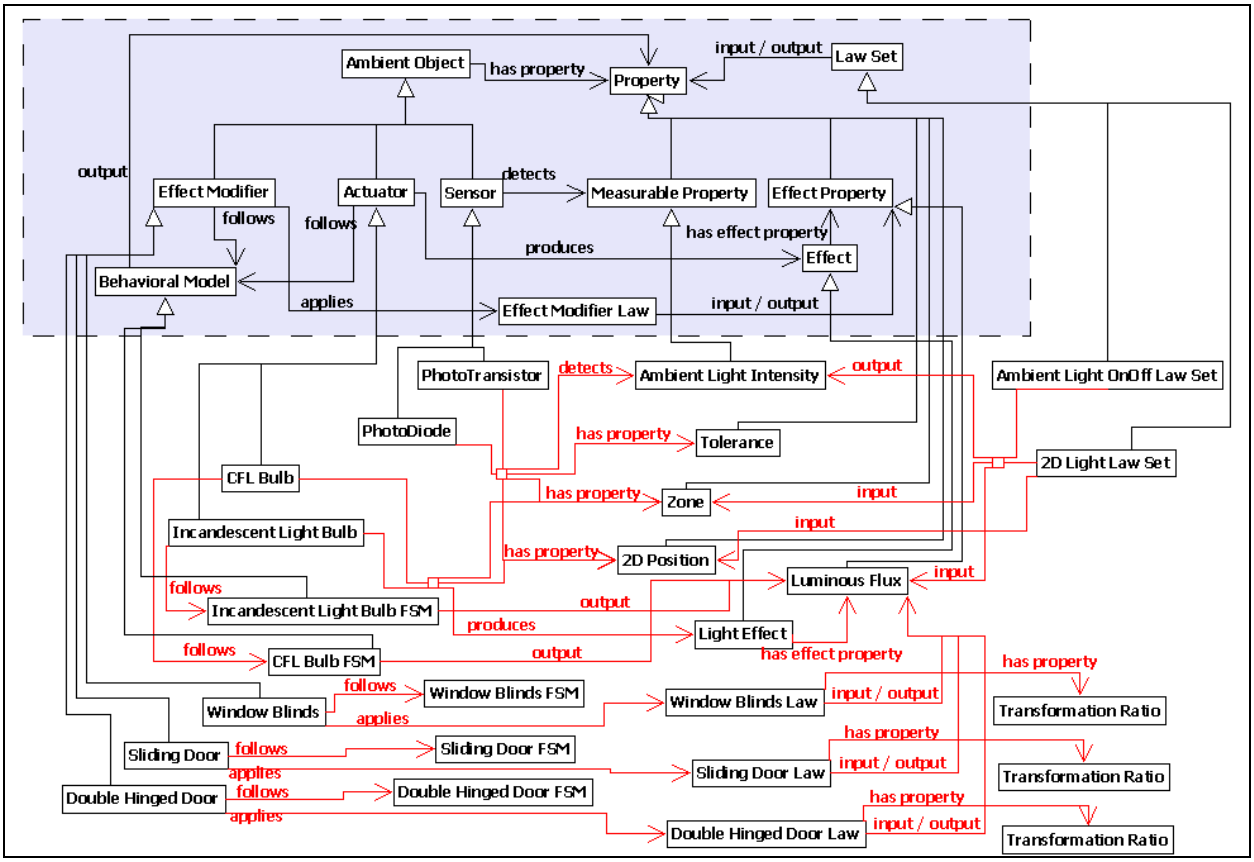

Figure 110. Concrete Model (down) created from the Abstract Model (top) in the context of Light FDD

Following the general structure of the abstract environment model, light actuators produce an effect, namely a light effect that has the effect property *Luminous Flux*. The value of the luminous flux is different for each light actuator, and it is determined by the characteristics of each type of the light actuator and its current state as described in the behavioral model. Light sensors detect the measurable property *Ambient Light Intensity*. The latter property is going to be the basis for fault detection, since it is going to be calculated via the prediction model and compared to the real value read by the sensor.

6.3.2.2. The mathematical model

To predict the theoretical value of the ambient light intensity, a set of laws is defined. The law set is composed of mathematical functions that are used in real time to perform calculations.

In input, the mathematical functions use the values of properties from instances and results from other functions within the same law set. One, and only one, mathematical function within a law set must have as output the measurable property measured by the sensor, in our case the ambient light intensity. Here is a reminder of the ambient light law set mathematical functions for 2 dimensional described spaces:

$$\text{sameZone}(o,a) = zone(o) \equiv zone(a) \tag{L1}$$

$$\text{distance}(o,a) = \begin{cases} \sqrt{(x(o)-x(a))^2 + (y(o)-y(a))^2} & when\ \text{sameZone}(o,a)\ is\ true \\ +\infty \quad when\ \text{sameZone}(o,a)\ is\ false \end{cases} \tag{L5}$$

$$\text{directLightExposure}(o,a) = \frac{\text{luminousFlux}(a)}{\text{distance}(o,a)^2} \tag{L3}$$

$$\text{ambientLightIntensity}(o) = \sum_{a \in LightEmitters \setminus o} \text{directLightExposure}(o,a) \tag{L4*}$$

Note that we use the variable name **o**: for object, instead of **s**: for sensor, since in this case sensors and Effect Modifiers will use these laws to estimate the amount of light intensity they are exposed to at their respective positions.

(L1) verifies whether or not an object and an actuator are in the same zone. For instance, when called for **(1.)**, and **(.5)** from **Figure 109**, ***(L1)*** returns **true** since zone of **(1.)** and zone of **(5.)** is ***room 1***. Zone here is an input. It is to be noted here that in order for the prediction engine to perform the calculations correctly, the model is completed with additional information in the form of properties to the declared types. At the instance level we find the actual values of these properties. These values can be inherited from the types, like in the case of tolerances (declared as a property of the type light sensor), or updated via the context engine, such as the positioning of tracked mobile components (for instance components equipped with Ubisence tracking system **[212]**). For non mobile objects, the coordinates can be added manually by the designer (or the final user "human diagnoser") at runtime.

(L2) uses the x,y coordinates (when they are defined) to calculate the distance between an actuator and any object that are in the same zone, we make a choice to return the an infinite distance value when the two objects are not in the same room.

(L3) estimates the light intensity value at the current position of an object when exposed to a single light source that is positioned at a certain distance (calculated from ***(L2)***) and that is generating a certain luminous flux. The input parameter *luminous flux* is the effect property that ensures that ***(L3)***, and consequently the whole ambient light law set, only considers actuators that produce light effect.

(L4*) calculates the sum of all the results from ***(L3)***, which is the sum of the light intensities caused by each single light source on this particular object. ***(L4*)*** is the function that calculates the theoretical value of the measurable property ambient light intensity around a Sensor (to be compared to the actual reading), and an Effect Modifier (to be transmitted). Note that this function considers all light emitters (i.e Actuators and Effect Modifiers) except the object that is calling the function. This avoids infinite loop when an Effect Modifier calls the function.

As defined in section **4.5**, Effect Modifiers are active components that do not produce an effect directly. However they do let a part of the effect circulate from one zone to another. Regardless of how small the interference is in some cases, it is important to model it and consider it in calculations in order to have more accurate fault detection and diagnosis results. For that transformation laws are defined for every Effect Modifier. These formulas calculate how much of the Effect that an Effect Modifier is exposed to in a zone (using ***L4****, ***L3***, ***L2***, and ***L1***), is transmitted to the other zone.

For example to calculate how ***room 1*** affect ***room 2*** in **Figure 109**. First, the ambient light intensity is calculated around the position of the double-hinged door (the Effect Modifier between room 1 and room 2) as if it was a sensor (the ambient light law set can be reused in this

case). Then, the Effect Modifier Law (from section **4.5**) is used to convert the calculated *light intensity* value into *luminous flux* again. It is in this conversion, from ambient light intensity to luminous flux, that the Effect Modifier property *transformation ratio* is considered in the calculations. The value of this transformation ratio is governed by the current state of the Effect Modifier defined in its behavioral model. Now in ***room 2***, the Effect Modifier is considered as an actuator generating the newly calculated luminous flux. This flux is considered when estimating ambient light intensity around a light sensor in this room. This approach results in much accurate estimation of the sensor's reading.

To sum up, we can say that an Effect Modifiers behaves like a sensors in the sense that it uses the law set to estimate the light it is exposed to, and it behaves like an actuator in the sense that it produces an Effect which property value is deduced from the calculations done as a sensor multiplied by a transformation ratio.

So when Light Modifiers need to estimate the value of the Light Intensity they are exposed to, in order for them to estimate the amount they let through to a neighboring area, they need to call ***(L4*)***. The latter function ignores the contribution of other Light Modifiers in the calculations.

When an Effect Modifier **Em** behaves as an actuator in the current zone, the law that calculated the value of the emitted luminous flux for **Em** is:

$$\texttt{LuminousFlux}_{(\texttt{Em})} : \Phi = \gamma \,.\, I_{(Em)} \qquad (\textbf{\textit{L5}})$$

With γ the transformation ratio defined by the **Em** Behavioral Model.

And I(Em) is estimated by calling ***(L4*)***.

6.3.2.3. **The Behavioral Models**

From the structure of the Abstract Model, and as shown in the description, every type of active objects (those who affect the environment in anyway; in our case: light actuators and Light Effect Modifiers) has a well defined behavior. To describe this behavior in a formal approach we use finite state machines. At instance level, the final values of every instance property depend on the current object state. Note that all objects have to be described at least according to the minimal state machine, which is any equivalent of the on-off state machine with two transitions equivalent to turn on-turn off.

So, at instance level, the Incandescent Light Bulb **(1.)** (from **Figure 109**) would have the finite state machine described in **Figure 66** with value_1 corresponding to *luminous flux* **Φ**, with a value equal to 1750 lm. And the incandescent Light Bulb **(3.)** would have **Φ** equal to 800 lm.

As for the CFL Light Bulb **(2.)**, it would have the behavioral model defined in **Figure 68**. The values of the properties such as the incremental increase of light flux with time "i", the warm up time "α", the cooling up time "β", and the flux value "Φ" are all deduced from the object characteristics at the instance level.

The behavior of the double hinged door as described in the environment description paragraph is modeled in the finite state machine in **Figure 111**. The value of the transformation rate from luminous intensity to luminous flux γ is updated with every transition. So at instance level the value of γ is deduced from the current state.

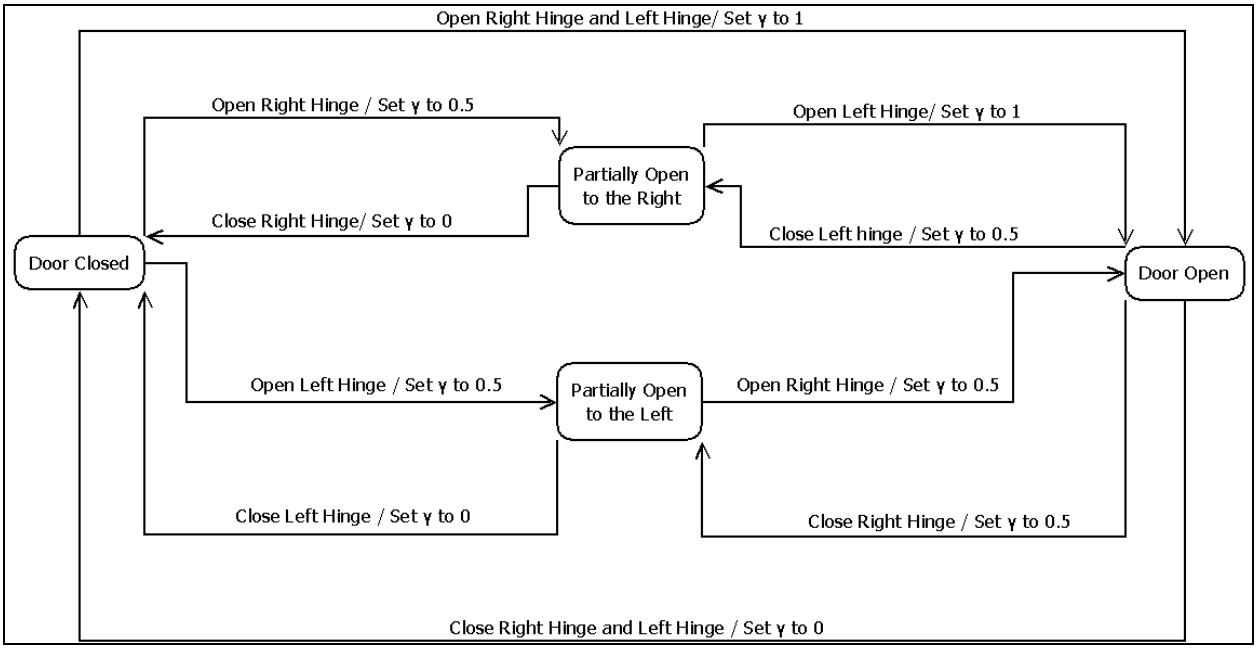

Figure 111. Double hinged Door Finite State Machine

The window blinds finite state machine is depicted in **Figure 112**.

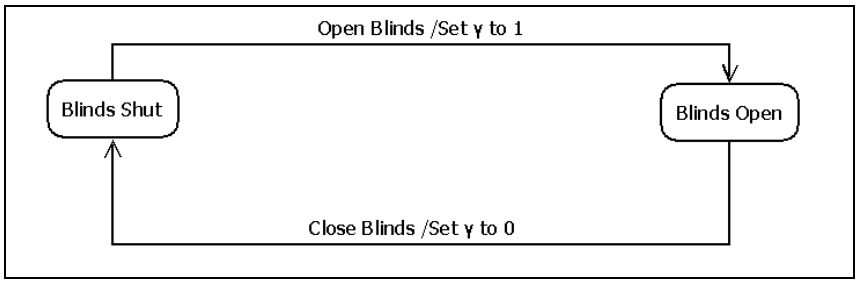

Figure 112. Window Blinds Finite State Machine

The finite state machine of the sliding door is in **Figure 113**.

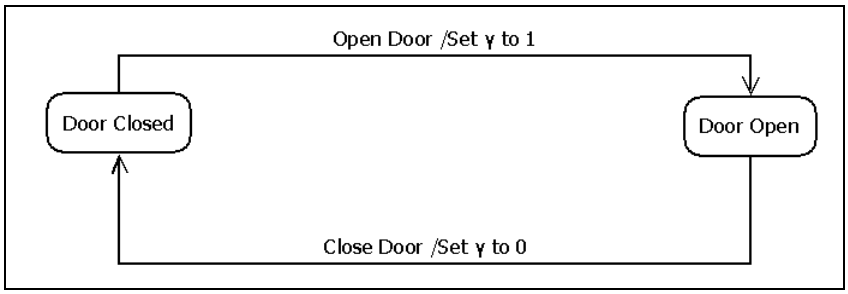

Figure 113. Sliding Door Finite State Machine

6.3.3. Instantiating the Models

Following the description of the Ambient Environment in **Figure 109**, we create the instances from the types created in the *concrete model* as follows:

Room 1: instance of the type *Zone*.

Room 2: instance of the type *Zone*.

Outside: instance of type *Zone*.

Light Bulb1: instance of the type *Incandescent Light Bulb*.

Light Bulb 2: instance of *Incandescent Light Bulb*.

Light Bulb 3: instance of *CFL Bulb*.

Light Sensor 1: instance of the type *Photo Diode*.

Light Sensor 2: instance of the type *Photo Diode*.

Light Sensor 3: instance of the type *Photo Transistor*.

Interior Door: instance of *Double Hinged Door*.

Main Window: instance of *Window Blinds*.

Main Door: instance of *Sliding Door*.

6.3.4. Performing fault detection

6.3.4.1. The simulator and building the Prediction Model

We define the Concrete Model and the instances using the triples. This definition is used by the simulator to build the Prediction Model.

First the definition of the Concrete Model (as detailed in **Figure 110**):

```
LightActuator TYPE Actuator;
FluorescentLightBulb TYPE LightActuator;
IncandescentLightBulb TYPE LightActuator;

PhotoDiode TYPE Sensor;
PhotoTransistor TYPE Sensor;

DoubleHingedDoor TYPE Modifier;
SlidingDoor TYPE Modifier;
WindowBlinds TYPE Modifier;

AmbientLightModificationRatio TYPE PhysicalProperty;
DoubleHingedDoor hasProperty AmbientLightModificationRatio;
SlidingDoor hasProperty AmbientLightModificationRatio;
WindowBlinds hasProperty AmbientLightModificationRatio;

ModifierLawSet2D TYPE LawSet;

ModifierLaw2D TYPE Function;
ModifierLaw2D hasOutput LightFlux;
ModifierLaw2D hasExpression
modifier.AmbientLightModificationRatio*TotalIllumination2D(modifier);
ModifierLaw2D hasArg modifier;

ModifierLawSet2D hasFunction ModifierLaw2D;

ModifierLawSetReleyEffect TYPE LawSet;

ModifierLawReleyEffect TYPE Function;
ModifierLawReleyEffect hasOutput LightFlux;
ModifierLawReleyEffect hasExpression
       modifier.AmbientLightModificationRatio*sensor.AmbientLight;
ModifierLawReleyEffect hasArg sensor;
```

```
ModifierLawSetReleyEffect hasFunction ModifierLawReleyEffect;

AmbientLight TYPE PhysicalProperty;

LightFlux TYPE PhysicalProperty;

X TYPE AbsolutePosition;
Y TYPE AbsolutePosition;
Zone TYPE Area;

FluorescentLightBulb produces LightFlux;
IncandescentLightBulb produces LightFlux;

PhotoDiode detects AmbientLight;
PhotoTransistor detects AmbientLight;

FluorescentLightBulb hasProperty X;
FluorescentLightBulb hasProperty Y;
FluorescentLightBulb hasProperty Zone;

IncandescentLightBulb hasProperty X;
IncandescentLightBulb hasProperty Y;
IncandescentLightBulb hasProperty Zone;

PhotoDiode hasProperty X;
PhotoDiode hasProperty Y;
PhotoDiode hasProperty Zone;

PhotoTransistor hasProperty X;
PhotoTransistor hasProperty Y;
PhotoTransistor hasProperty Zone;

LightEffect TYPE Effect;

AmbientLightLawSet2D TYPE LawSet;

   #ambientLightIntensity function is called here TotalIllumination2D
   #to be compatible with corresponding java function.
   #Different java function were defined for 3D, 2D, and onOff
   #similar redefining was done for the other functions of the Law-Sets

TotalIllumination2D TYPE Function;
TotalIllumination2D hasOutput AmbientLight;
TotalIllumination2D hasExpression _SUM(a, LightActuator, SingleIllumination2D(a,
      sensor));
TotalIllumination2D hasArg sensor;

AmbientLightLawSet2D hasFunction TotalIllumination2D;

SingleIllumination2D TYPE Function;
SingleIllumination2D hasOutput singleSourceLight;
SingleIllumination2D hasExpression
      (SameZone(actuator,sensor)*actuator.LightFlux)/(Distance2D(actuator,
sensor)^2);
SingleIllumination2D hasArg actuator;
SingleIllumination2D hasArg sensor;

AmbientLightLawSet2D hasFunction SingleIllumination2D;

Distance2D TYPE Function;
Distance2D hasOutput distance;
Distance2D hasExpression (((a.X-b.X)^2)+((a.Y-b.Y)^2))^(0.5);
Distance2D hasArg a;
Distance2D hasArg b;

AmbientLightLawSet2D hasFunction Distance2D;

SameZone TYPE Function;
```

```
SameZone hasOutput sameZone;
SameZone hasExpression (0^(a.Zone-b.Zone));
SameZone hasArg a;
SameZone hasArg b;

AmbientLightLawSet2D hasFunction SameZone;

LightEffect hasLawSet AmbientLightLawSet2D;

AmbientLightLawSetOnOff TYPE LawSet;

TotalIlluminationOnOff TYPE Function;
TotalIlluminationOnOff hasOutput AmbientLight;
TotalIlluminationOnOff hasExpression _OR(a, LightActuator,
SingleIlluminationOnOff(a, sensor));
TotalIlluminationOnOff hasArg sensor;

AmbientLightLawSetOnOff hasFunction TotalIlluminationOnOff;

SingleIlluminationOnOff TYPE Function;
SingleIlluminationOnOff hasOutput SingleSourceLight;
SingleIlluminationOnOff hasExpression SameZone(actuator,sensor)*actuator.LightFlux;
SingleIlluminationOnOff hasArg actuator;
SingleIlluminationOnOff hasArg sensor;

AmbientLightLawSetOnOff hasFunction SingleIlluminationOnOff;

AmbientLightLawSetOnOff hasFunction SameZone;

LightEffect hasLawSet AmbientLightLawSetOnOff;

FluorescentLightBulb FSM resources/FSM/FluorescentLightBulb_FSM.xml;
IncandescentLightBulb FSM resources/FSM/IncandescentLightBulb_FSM.xml;
DoubleHingedDoor FSM resources/FSM/DoubleHingedDoor_FSM.xml;
SlidingDoor FSM resources/FSM/SlidingDoor_FSM.xml;
WindowBlinds FSM resources/FSM/WindowBlinds_FSM.xml;
```

Then the declaration of the instances:

```
#room1: Zone = 1
#room2: Zone = 2
#outside: Zone = 3

bulb1 is IncandescentLightBulb;
bulb1 X 1;
bulb1 Y 2;
bulb1 Zone 1;
bulb1 LightFlux 1750;
bulb2 is FluorescentLightBulb;
bulb2 X 3;
bulb2 Y 2;
bulb2 Zone 2;
bulb2 LightFlux 1500;
bulb3 is IncandescentLightBulb800;
bulb3 X 4;
bulb3 Y 4;
bulb3 Zone 2;
bulb3 LightFlux 800;
sensor1 is PhotoDiode;
sensor1 X 3;
sensor1 Y 4;
sensor1 Zone 2;
sensor1 AmbientLight 20;
sensor2 is PhotoDiode;
sensor2 X 1;
```

```
sensor2 Y 4;
sensor2 Zone 1;
sensor2 AmbientLight 25;
sensor3 is PhotoTransistor;
sensor3 Zone 3;
sensor3 AmbientLight 33000;
doubleHingedDoor is DoubleHingedDoor;
doubleHingedDoor X 2;
doubleHingedDoor Y 2;
doubleHingedDoor Zone 1;
doubleHingedDoor Zone 2;
doubleHingedDoor AmbientLightModificationRatio 0;
slidingDoor is SlidingDoor;
slidingDoor X 1;
slidingDoor Y 0;
slidingDoor Zone 1;
slidingDoor Zone 3;
slidingDoor AmbientLightModificationRatio 0;
slidingDoor TrustedSensor sensor3;
windowBlinds is WindowBlinds;
windowBlinds X 0;
windowBlinds Y 3;
windowBlinds Zone 1;
windowBlinds Zone 3;
windowBlinds AmbientLightModificationRatio 0;
windowBlinds TrustedSensor sensor3;
```

In order for the 'same zone' law to be better interpreted by the Mathematical Model we convert it to a mathematical power expression **`(0^(a.Zone-b.Zone))`**, which will return **1** only if a and b Zones are equal **`(0^0=1)`**, and **0** otherwise. For that we did not instantiate room1 and room2 as instances of 'Zone', instead we directly assign an integer value representing the zone of each component. The mapping between zones and the integer values is given as comment before the instance section.

Note that we define the initial value for **sensor3** (33000 lux) which is an estimate of the direct sun exposure Illuminance (usually between 32000 lux and 130000 lux. See **Table 4**). Being a special case of a sensor reporting the Illuminance outside, this value will be used for calculating the amount of Light Intensity that is passed through Effect Modifiers connecting room 1 with the outside.

6.3.4.2. **The simulation**

Using this triplet definition of our environment we obtain a Prediction Model (depicted in **Figure 115**) that contains, in addition to finite state machines to control the Actuators' behavior, finite state machines that controls the Modifiers' behavior. The control panel for the controllable Properties is depicted in **Figure 114**

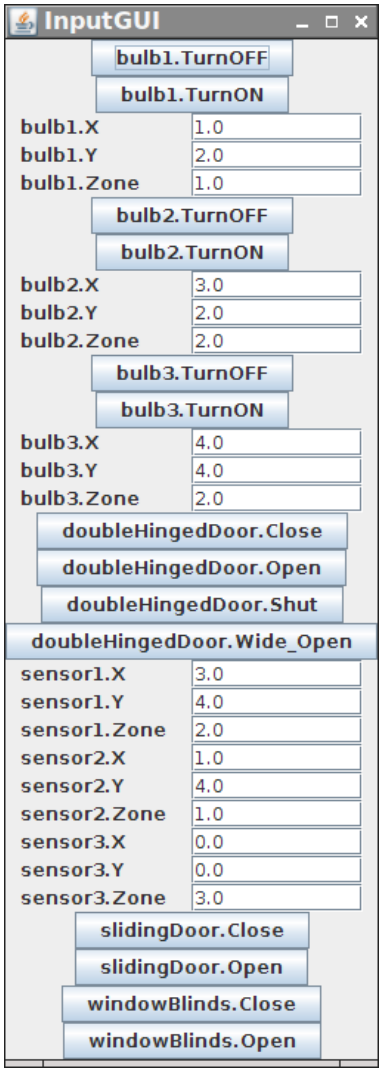

Figure 114. Control panel for the Prediction Model inputs – scenario 2

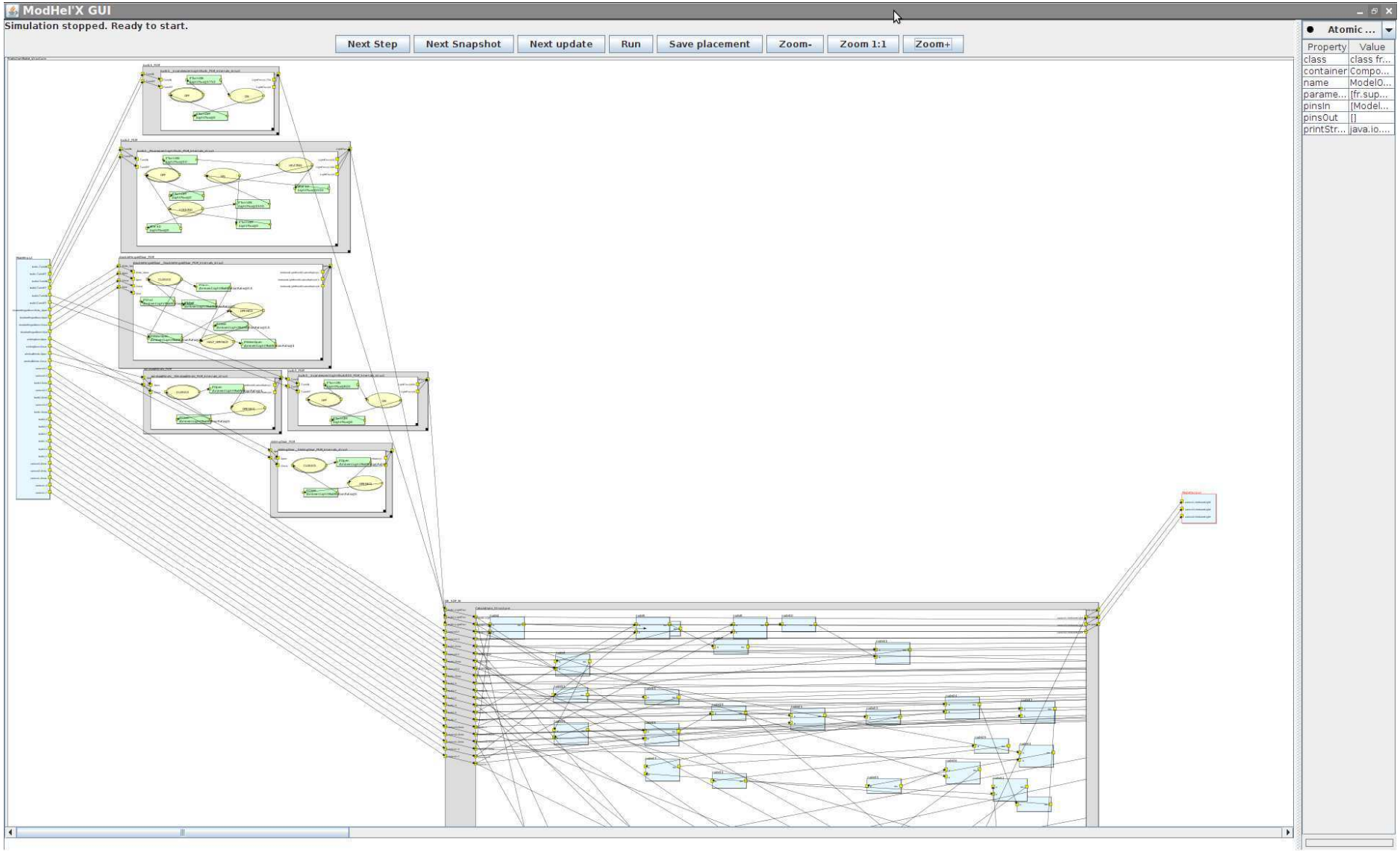

Figure 115. General View of the ModHel'X representation of the Prediction Model for Scenario 2

We start the simulation with all finite state machines at their initial states (all Actuators are off and all Modifiers are closed), and we obtain the output depicted in **Figure 116**. Then we run a series of test consisting in specific situations described in the Prediction Model by a certain combinations of states.

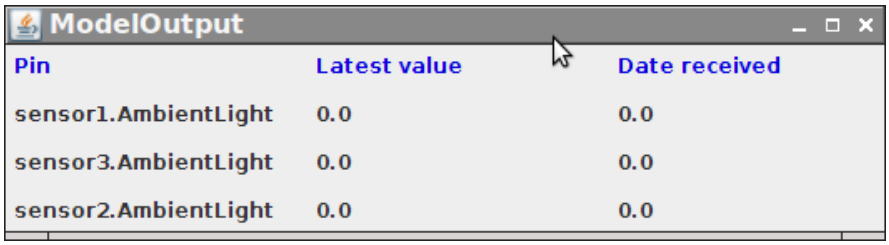

Figure 116. Scenario_2 simulation trace values – Initial values

We focus in this scenario on the consequences of introducing Modifiers to the Fault Detection results. However we start by verifying that turning on all Actuators produce the expected results in each separate room (we keep the Effect Modifiers closed for now).

Calculation verifications for all test results in Scenario_2 are in Annex-C.

Test 1: We turn on **bulb1** (emitting light flux Φ = 1750 lm), **bulb2** (Φ = 1500 lm), and **bulb3** Φ = 800 lm) and we observe the changes to the predicted sensors' readings:

As expected (from turning on bulb_2, which is a CFL light bulb) we observe a transition phase (see **sensor1** reading in **Figure 117**) where bulb2 is heating, before attaining its full luminosity (see **sensor1** reading in **Figure 118**).

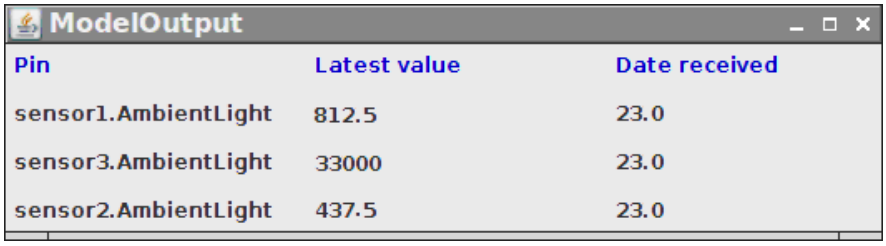

Figure 117. Scenario_2 Test1 simulation trace values – (1/2)

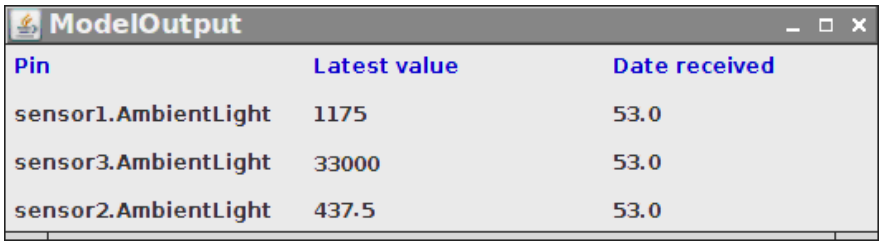

Figure 118. Scenario_2 Test1 simulation trace values – (2/2)

So when all light bulbs are On, and all effect modifiers are closed we have **1175** lux read by **sensor1**, and **437.5** lux read by **sensor2**.

Test 2: Keeping all the light bulbs On, we open wide (the two hinges of) the double hinged door, thus allowing (by applying ***L5*** with transformation ratio property of the double hinged door equals to **1**) the propagation of the full amount of light that reaches the door from one room to the other (in the two directions). This amount of light at the double hinged door is calculated using the ambient light law set mathematical functions for 2 dimensional described spaces as if the double hinged door was a light sensor. The resulting value is then fully transmitted (as the transformation ratio is 1) to the neighboring zone in the form of light flux as if the double hinged

door was a light actuator. This is done in two directions: from room1 to room2 and from room2 to room1.

The definition of ***L5*** in the triplet syntax defining the concrete model is (note the call to ***L4*** with the modifier as argument instead of a sensor):

```
modifier.AmbientLightModificationRatio * ambientLightIntensity(modifier);
```

As expected, an augmentation in the readings of **sensor1** and **sensor2** is noticed (**Figure 119**):

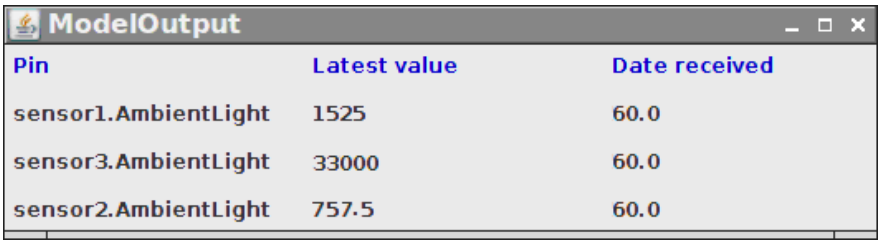

Figure 119. Scenario_2 Test2 simulation trace values (1/2)

When we opened the double hinged door, **sensor1**(in room2) became exposed to light (from room1) passing through the Double Hinged Door, and we notice an augmentation in reading value for **sensor1** from **1175** to **1525** lux. Likewise, when the double hinged door was opened, **sensor2**(in room1) became exposed to light (from room2) passing through the Double Hinged Door, and we notice an augmentation in reading value for **sensor2** from **437.5** to **757.5** lux.

Then we close the double hinged door and we observe the same values as in **Figure 118** (readings when all modifiers were closed and all light bulbs are On).

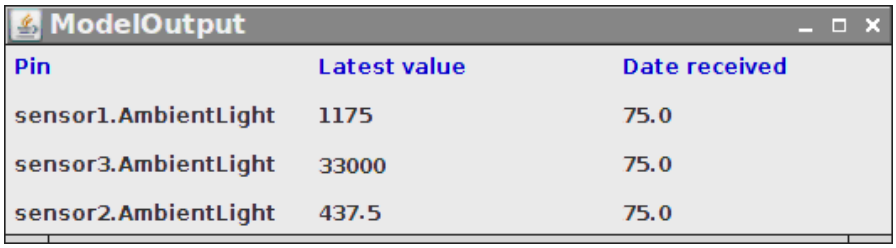

Figure 120.Scenario_2 Test2 simulation trace values (2/2)

Test 3 (special case of relaying an external sensor value): For this special case we create the Law-Set (we calle `ModifierLawSetReleyEffect` in the triplet syntax in the Concrete Model) that uses the value of readings of a sensor, instead of calculating the amount of light received by a modifier using Ambient Light Law-Set. To the value of the reading of the sensor we apply the modifier's transformation ratio. The definition of this special law is:

```
modifier.AmbientLightModificationRatio * sensor.AmbientLight;
```

Note that, at the contrary of other laws where instances to be used in the Prediction Model are deduced at run-time (hence the actuator-sensor decoupling aspect of our framework), `sensor` here is explicitly linked to the modifiers it provide readings to. The triplets defining these links are (at instance level):

```
windowBlinds TrustedSensor sensor3;
slidingDoor TrustedSensor sensor3;
```

We keep the all the light bulbs On, and the double hinged door and the sliding door closed, then we change the state of window blinds (connecting room1 to the outside) to 'Open', thus

making it relay external Ambient Light that is read by sensor3. We should not observe any changes in the value of **sensor1** (**1175** lux) in room2, however we should observe an augmentation in the readings of **sensor2** in room1.

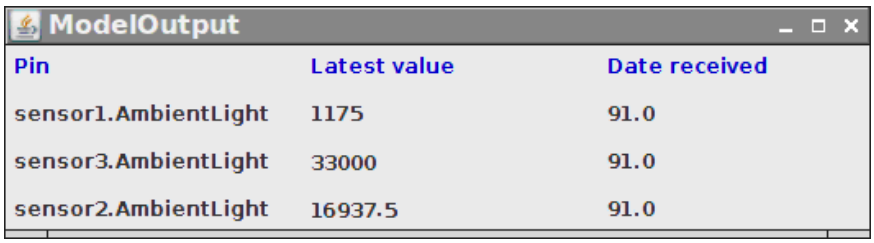

Figure 121. Scenario2 Test3 simulation trace values

As expected, we observe an augmentation in the readings of sensor2 in room1 (**Figure 121**) from **437.5** to **16937.5**.

Test 4 (continuing the special case of relaying an external sensor value – Effect Modifier relaying effect of another Effect Modifier):

Keeping the windowBlinds open, we then open one of the hinges of the doubleHingedDoor. We should observe an augmentation of the readings of sensor1 in room2 now that it is exposed to some (which should be half of the light intensity that reaches the doubleHingedDoor according to its finite state machine) of the light of room1, which in turn is exposed to external light coming from the outside though the windowBlinds.

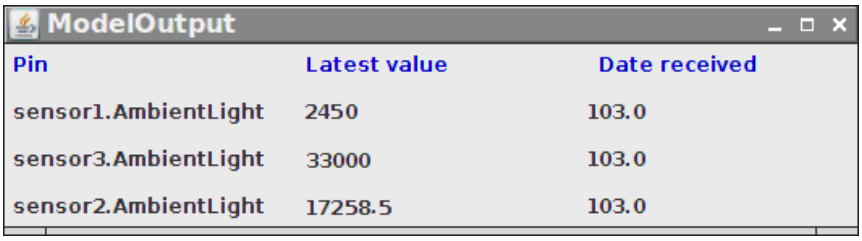

Figure 122. Scenario2 Test4 simulation trace values

As expected, we observe an augmentation in the readings of **sensor1** in **room2** (**Figure 122**) from **1175** to **2450**, and in the readings of **sensor2** in **room1** from **16937.5** to **17258.5**. The latter augmentation is caused by the (half of) light of **bulb2** and **bulb3** passing from room2 to room1. In fact, when the doubleHingedDoor calculated to contribution in light from **room2** to **room1**, it only considered **bulb2** and **bulb3** in the calculations, ignoring by that its own contribution to light intensity in **room2**; that is done in order to avoid an infinite loop. Hence only a 320 lux augmentation (**17258.5**=16937.5+320) is noticed by **sensor2**, which is the same augmentation observed earlier (in **test_2**) when the doubleHingedDoor was opened, while the windowBlinds were closed (757.5=437.5+320).

6.4. Scenario_3: Ambient Bathtub fault detection and diagnosis (Heat Effect and Water Flow Effect):

In this example, we will see how fault detection and diagnosis is performed in a scenario where a bathtub is being filled. In this scenario we combine two different Effects: Heat Effect and Liquid Flow Effect, which is why we have non typical Bathtub that behaves like a hot tub in the sense that it has a resistor allowing heating the water in the tub. This allows us to apply the same approach of fault detection to the Heating System of an Ambient Room for instance. All it

is needed to do is to change the Heat Effect carrier from water to air, which imposes the changing of a single constant value (volumetric heat capacity; se **Annex-B**).

As illustrated in **Figure 123**, we have a bathtub and four actuators controlled by the system's controller:

- A hot water tap: an actuator controlled by the system for pouring hot water at the rate of 110cm^3/s.
- A cold water tap: for cold water, pouring cold water at the rate of 140cm^3/s.
- A water drain: an actuator controlled by the system for releasing water, at the rate of 50 cm^3/s.
- A resistor: a "kettle element" (like that found in electric kettle) actuator for controlling the temperature of the water in the bathtub, generating 2.5kW of power.

There are also two sensors reporting the state of the environment to the Ambient System (water temperature and level):

- A thermometer: reporting water temperature in Celsius.
- A liquid level indicator: reporting water level in the bathtub. Used to estimate water quantity in the bathtub.

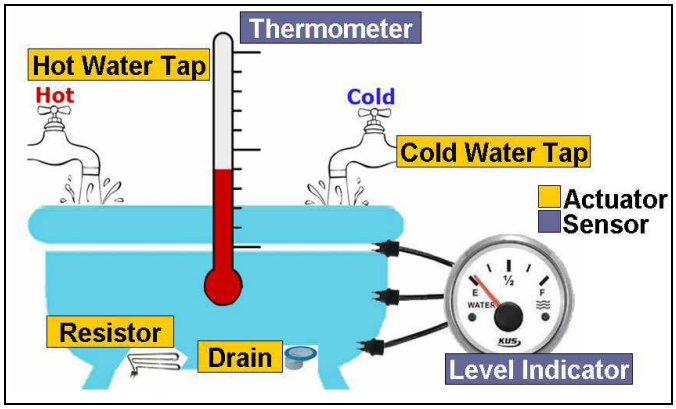

Figure 123. Components of the Bathtub Fault Detection and Diagnosis Example

6.4.1. Building the models

6.4.1.1. The concrete model

As in previous scenarios we start by building the environment models. The models will be the basis for creating the proper instances representing the components described in **Figure 123**.

We start by identifying the different types of actuators and sensors. In order to do that we identify what effects are produced by the actuators and what physical properties are detected by the sensors. In fact the water taps (cold and hot) and the drain can be classified as **Liquid Discharge Actuators** (as described in **4.2.3**), with positive (or null) **Discharge Rate** for the water taps, and negative (or null) Discharge Rate for the water drain. The level indicator is the **Liquid Level Sensor**. These Actuators and Sensors are going to be linked at run-time by the Prediction Model using the **Liquid Flow Effect**. The Resistor and the thermometer are, respectively, the **Heat Actuator** and the **Heat Sensor** (as described in **4.2.2**) in this scenario. The Concrete Model is depicted in **Figure 124**.

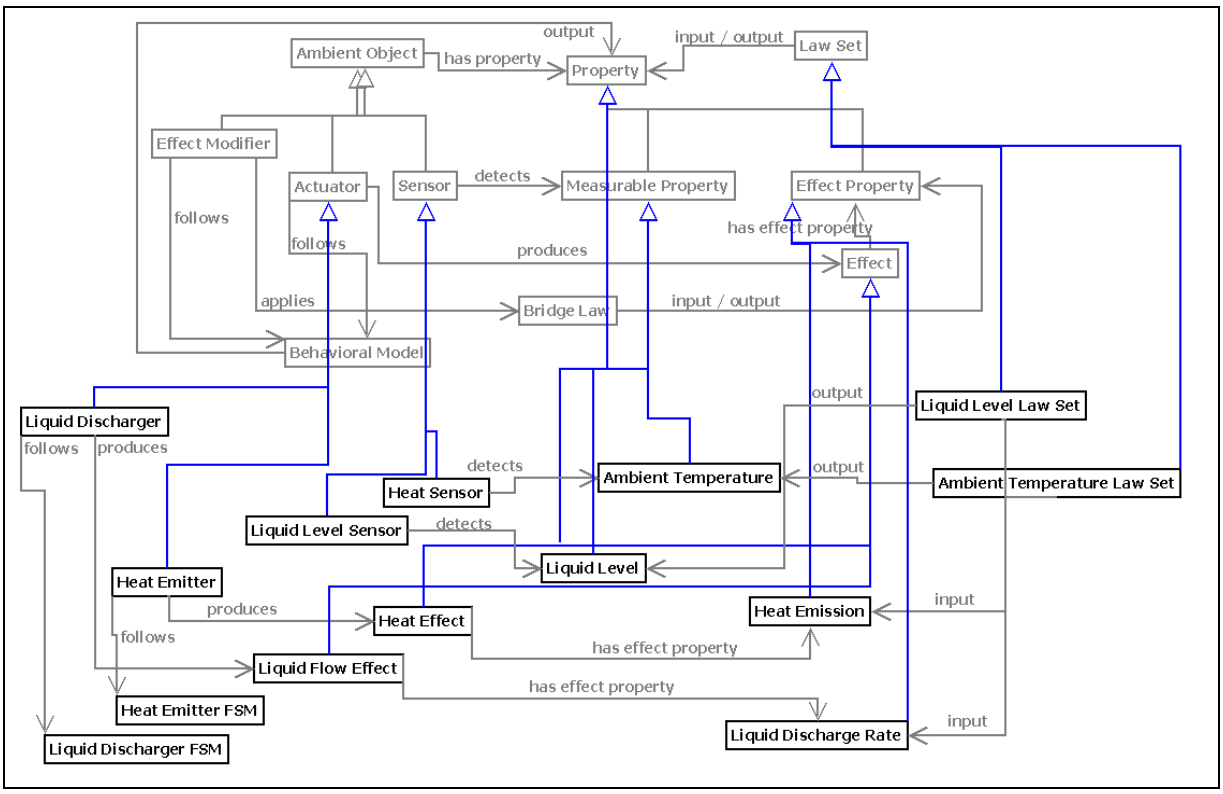

Figure 124. The Concrete Model for the Bathtub Fault Detection and Diagnosis

6.4.1.2. The mathematical models

To predict the theoretical value of the Liquid Level Indicator and the Thermometer, we use the *Liquid Level Law Set* (defined in **4.2.3.1**), but with removing **L1** (same zone verification law) since we only have one zone (the bathtub) to consider in this scenario. This gives us the following law-set:

$$level(s,a) = \text{dischargeRate}(a) \times elapsedTime \qquad \textbf{(L12')}$$

$$totalLevel(s) = l_0 + \sum_{a \in WaterSources} level(s,a) \qquad \textbf{(L13)}$$

(L12') has the same mathematical formula as **L12** but without the 'same zone' condition.

To predict the theoretical value of the water Temperature we use the *Ambient Temperature Law Set* (as described in **4.2.2.2**). Here also we ignore the 'same zone' verification law. This gives us the following law-set:

$$\text{Energy}(s,a) = heatEmission(a) \times elapsedTime \qquad \textbf{(L8')}$$

$$\text{temperature}(s) = \left(\sum_{a \in HeatActuators} Energy(s,a) \right) \div (v \times c) \qquad \textbf{(L9)}$$

With:

c is the volumetric heat capacity for water. The volumetric heat capacity of water is 4.1796 $J.cm^{-3}.K^{-1}$. (See **Annex-B** for the complete list of volumetric heat capacities of different chemicals).

v is the total volume of the water. This value (in cm^3) can be deduced via ***L13*** from the *Liquid Level Law Set*.

(*L8'*) has the same mathematical formula as **L8** but without the 'same zone' condition.

6.4.1.3. The behavioral models

As in previous scenarios we use finite state machines as our behavioral models.

- **The water taps** (Liquid Discharger) have two possible states:

  - Idle, in which case they don't discharge water (Discharge Rate equals to 0);

  - Discharging Water, in which case they discharge a fixed amount **d** of water in liter per second. That amount is specific to each instance of Water Tap.

  The finite state machine for Water Taps is depicted in **Figure 125**.

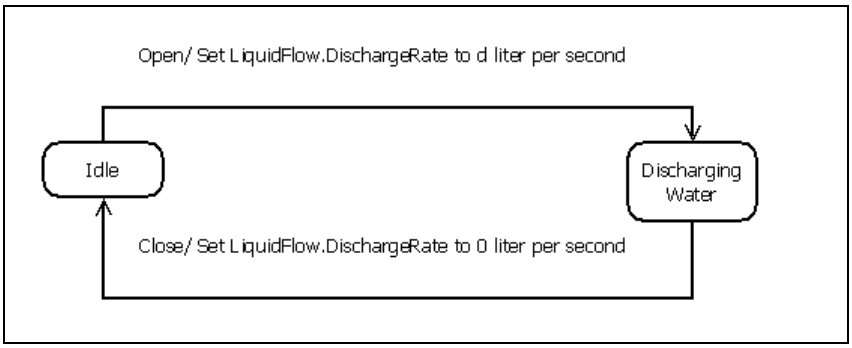

Figure 125. Water Discharger finite state machine

- **The bathtub drain** (Water Evacuator) can be:

  - Opened, in which case it has a value **d** of discharge rate. This value should be negative;

  - Closed, in which case it has a null value of discharge rate.

  The finite state machine of the bathtub drain is described in **Figure 126**.

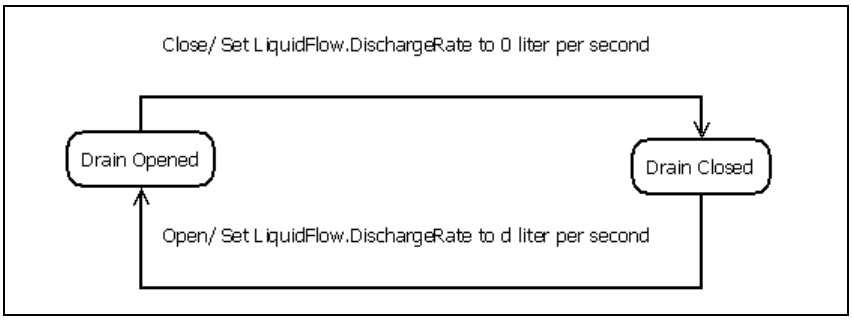

Figure 126. Water Discharger finite state machine

We note here that the finite state machines of the bathtub taps and drain can be grouped into one state machine with two global states (Inactive, Active). In which case when transitioning from **Inactive to Active**, the value of discharge rate would be set to a positive value when the

FSM is instantiated by a water tap, and to a negative value when it is instantiated by a water drain. The transition from **Active to Inactive** would always set the discharge rate to 0. However we adopt the solution of different FSMs for water taps and water drains because we reckon that having specific state names for each device type is a more explanatory solution.

- **The resistor** of the Bathtub also has two possible states:

  - Off, in which case it doesn't emit heat;

  - On, in which case it emits **h** joule per second of heat emission. **h** is also defined at instance level.

The finite state machine for the bathtub resistor is defined in **Figure 127**.

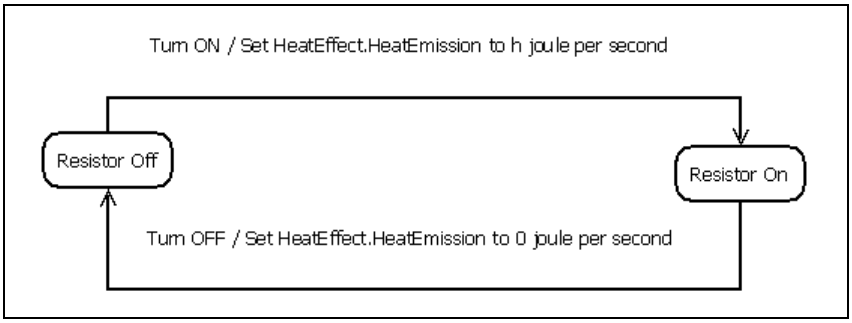

Figure 127. Resistor finite state machine

6.4.2. Instantiating the Models

Following the description of the Ambient Bathtub in **Figure 123**, we create the instances from the types created in the Concrete Model defined in **Figure 124** as follows:

Hot Water Tap: instance of the type *Liquid Discharger*, to which we associate a *Liquid Flow Effect*, having the property *Liquid Discharge Rate* with the value 110.

Cold Water Tap: instance of *Liquid Discharger*, to which we associate a *Liquid Flow Effect*, having the property *Liquid Discharge Rate* with the value 140.

Drain: instance of *Liquid Evacuator*, to which we associate a *Liquid Flow Effect*, having the property *Liquid Discharge Rate* with the value -50.

Bathtub Resistor: instance of the type *Resistor*.

Water Level Indicator: instance of the type *Liquid Level Sensor*.

Thermometer: instance of the type *Heat Sensor*.

We suppose that we have a constant *Water Discharge Rate* of 140cm^3 per second for Cold water and 110cm^3/s for Hot Water (when they are opened), and a constant *Water Discharge Rate* of 0 cm^3/s (drain is closed).

6.4.3. Performing fault detection

6.4.3.1. The simulator and building the Prediction Model

Here we define the Concrete Model and the instances using the triples' syntax.

First we use the triplet synthax for the definition of the Concrete Model that is illustrated in **Figure 124**:

```
LiquidDischarger TYPE Actuator;
LiquidEvacuator TYPE Actuator;
HeatEmitter TYPE Actuator;
WaterTap TYPE LiquidDischarger;
Drain TYPE LiquidDischarger;
Resistor TYPE HeatEmitter;

LiquidLevelSensor TYPE Sensor;
HeatSensor TYPE Sensor;

AmbientTemperature TYPE PhysicalProperty;
LiquidLevel TYPE PhysicalProperty;

HeatEmission TYPE PhysicalProperty;
LiquidDischargeRate TYPE PhysicalProperty;

WaterTap produces LiquidDischargeRate;
WaterTap produces LiquidDischargeRate;
Drain produces LiquidDischargeRate;
Resistor produces HeatEmission;

LiquidLevelSensor detects LiquidLevel;
HeatSensor detects AmbientTemperature;

LiquidFlowEffect TYPE Effect;
HeatEffect TYPE Effect;

LiquidLevelLawSet TYPE LawSet;

TotalLiquidLevel TYPE Function;
TotalLiquidLevel hasOutput LiquidLEvel;
TotalLiquidLevel hasExpression _SUM(a, LiquidDischarger, SingleDischarge(a));

LiquidLevelLawSet hasFunction TotalLiquidLevel;

SingleDischarge TYPE Function;
SingleDischarge hasOutput liquidLEvel;
SingleDischarge hasExpression (actuator.LiquidDischargeRate);
SingleDischarge hasArg actuator;

LiquidLevelLawSet hasFunction SingleDischarge;

LiquidFlowEffect hasLawSet LiquidLevelLawSet;

AmbientTemperatureLawSet TYPE LawSet;

TotalLiquidLevel TYPE Function;
TotalLiquidLevel hasOutput temperature;
TotalLiquidLevel hasExpression _SUM(a, HeatEmitter, SingleSourceHeat(a));

AmbientTemperatureLawSet hasFunction TotalLiquidLevel;

SingleSourceHeat TYPE Function;
SingleSourceHeat hasOutput energy;
SingleSourceHeat hasExpression a.HeatEmission*TIME*4.1796*TotalLiquidLevel(a);
SingleSourceHeat hasArg a;
```

```
AmbientTemperatureLawSet hasFunction SingleSourceHeat;

HeatEffect hasLawSet AmbientTemperatureLawSet;

WaterTap FSM resources/FSM/WaterTap_FSM.xml;
Drain FSM resources/FSM/Drain_FSM.xml;
Resistor FSM resources/FSM/Resistor_FSM.xml;
```

Then the declaration of the instances:

```
hotWaterTap is WaterTap;
hotWaterTap LiquidDischargeRate 0.14;
coldWaterTap is WaterTap;
coldWaterTap LiquidDischargeRate 0.11;
drain is Drain;
drain LiquidDischargeRate -0.05;
resistor is Resistor;
resistor HeatEmission 2500;
```

6.4.3.2. **The simulation**

When we run the simulator using the previous triplet definition of our environment we obtain the Prediction Model depicted in **Figure 129**. We can see the 4 finite state machines controlling the behavior of our 4 sensors in this scenario. We can change the state of these machines using the control panel as shown in **Figure 128**.

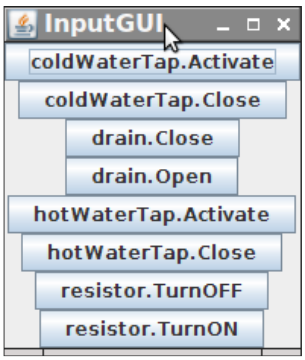

Figure 128. Control panel for finite state machines in scenario 3

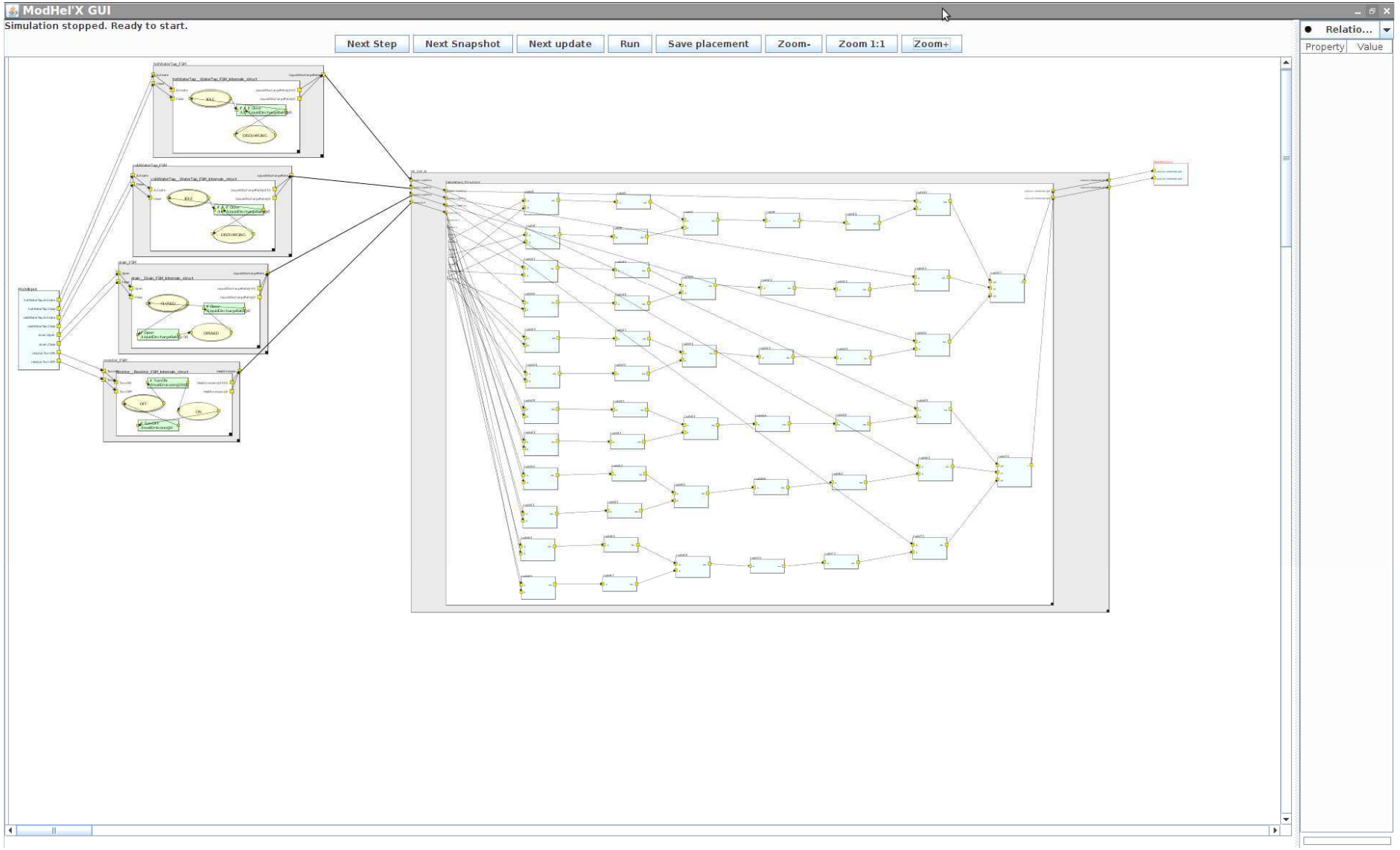

Figure 129. Prediction Model of the Bathtub scenario

In the next sections we verify the two observed Effects in this system separately.

Water Level test: (Liquid Flow Effect)

The fault detection task goes as follows: knowing the system's overall water discharge rate value (hot and cold water taps and the drainer), at any given time the FDD framework knows both the value of the water level detected by the level indicator sensor and the value of the expected water level calculated by the corresponding physical laws. Let us also suppose that diagnosis over water level is performed periodically every 3 seconds. **Table 1** illustrates the traces of the calculated water level every 30 seconds for 180 seconds ("timer"=0 being the moment the water taps are opened).

Time	Ambient Water Quantity *(From Prediction Model)*
0 s	0.00 cm^3 (0.00 liter)
30 s	7500.00 cm^3 (7.50 liter)
60 s	15000.00 cm^3 (15.00 liter)
90 s	22500.00 cm^3 (22.50 liter)
120 s	30000.00 cm^3 (30.00 liter)
150 s	37500.00 cm^3 (37.50 liter)
180 s	45000.00 cm^3 (45.00 liter)

Table 1. Water Level Expected Values 15 seconds after Water Taps are Opened

Calculation verification for the values at the second 150:

The water quantity predicted by the Water Flow Ambient Law is **37500cm^3**. It is the sum of the water quantity produced by the *Hot Water Tap* after 150 seconds (calculated by (L12')), the water quantity produced by the *Cold Water Tap* after 150 seconds (calculated by (L12')), and the water quantity evacuated by the *Drain* after 150 seconds (calculated by (L12')). The initial water level is set 0.

(L13) = (L12'):[140$cm^3.s^{-1}$ x 150s] + (L12'):[110$cm^3.s^{-1}$ x 150s] + (L12'):[0$cm^3.s^{-1}$ x 150s] + 0

The call tree representing the mathematical model of the predicted water level indicator is illustrated in **Figure 130**:

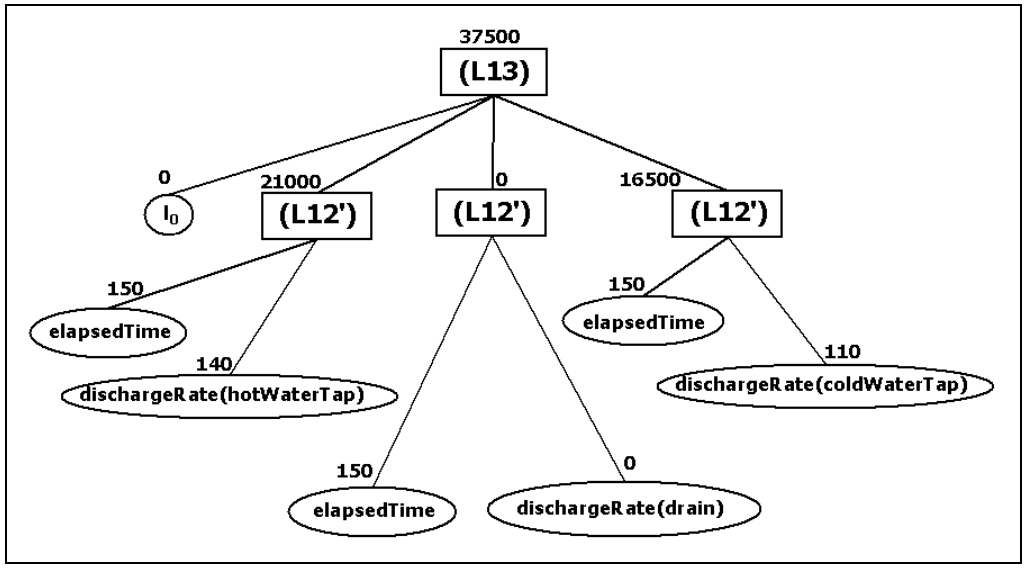

Figure 130. Scenario_3. Call tree for the Water Level Indicator at the second 150

Water Temperature test: (Heat Effect)

In this second part of the example, we verify the bathtub's "hot tub" functionality. Water already present in the bathtub is heated by an immersed heating element that is basically composed of a resistor that converts electric power into heat. We know that this heating element has a power rating of 2.5kW. We change the configuration so that water now comes only from the cold water tap. What we should notice here is that the water temperature elevation is incremental over time.

In order to estimate the water quantity in the bathtub (necessary for applying the Ambient Temperature Law-Set) we use the Liquid Level Law-Set. To remain consistent with previous results for water level calculations, we consider now that the cold water discharge rate is 250cm^3/s (which was previously the sum of hot and cold water discharge rates), and that the hot water tap is closed. With this configuration we obtain the same results for water level diagnosis as the first part of the example. We also consider that we have the property "heat power" (with the value of 2500J/s) as an effect property of the "heat emission effect" produced by the actuator "resistor". We also suppose that we have a constant loss of heat caused by the direct contact of the water with ambient air and the bathtub material, this heat loss is represented by a "heat power" of -500J/s; to differentiate from previous heat property we call this property "heat loss". As a total we then have a total "heat power" of 2000J/s produced by the combination of heat loss and the resistor.

During the first 3 minutes (180 seconds), we obtain the temperature traces illustrated in **Table 2**. The traces are calculated every 30 seconds.

The initial temperature variation is considered to be null "0 K", which verifies the calculations even though it is not realistic. This would result in the detection of a fault during the first seconds of the fault detection task. This can be overcome by defining an initial value to the water temperature. The problem would be to estimate such value. A possible solution would be adopting an approach inspired from the (special case) trusted external light sensor of the previous scenario. In which case, we would use the reading of **another** thermometer (when available) as an initial temperature value. It is very important to note that this requires the existence of another sensor, since using the reading of the same sensor would mean that we are using the reading of a sensor to estimate the value of readings the same sensor.

Calculation verification for the values at the second 150:

The water quantity calculated by the Water Flow Ambient Law is 37500cm^3

(L13)= (L12'):[250$cm^3.s^{-1}$ x 150s]+0+0+0.

The accumulated water heat energy, calculated by (L8'), is 300000J

(L8')= [2000$J.s^{-1}$x150s].

The ambient water temperature is calculated by (L9).

It is the result of the temperature augmentation at t=150s, which is 1.9140K

(L9)= [300000/(v.c); where v=37500cm^3; and c=4.1796 $J.cm^{-3}.K^{-1}$],

plus the temperature calculated at t=149s, which is equal to 285.1947K.

The final result is 285.1947+1.9140=287.1088K (13.95°C).

Time (s)	Water Quantity "Calculated Liquid Level Law-Set	Accumulated Water Heat Quantity "Calculated by (5)" (joule)	Ambient Water Temperature "Calculated by (6) and (7)" (K)
0	0	.	.
30	7500	60000	57.42 (-215.72°C)
60	15000	120000	114.84 (-158.30°C)
90	22500	180000	172.26 (-100.88°C)
120	30000	240000	229.68 (-43.46°C)
150	37500	300000	287.10 (13.95°C)
180	45000	360000	344.53 (71.38°C)

Table 2. Water Temperature Expected Values

6.5. Conclusion

In this chapter we ran some scenarios in order to test and validate our approach. In the first scenario we positioned our Fault Detection and Diagnosis approach in the context of an Ambient Assisted Living application. Even though our FDD framework had limited functionalities in that context (the Prediction Model generated was composed only of the Mathematical Model allowing the prediction of sensors values, but no description of devices behaviors was provided), we showed "how and when" our FDD framework would intervene in a real Ambient Environment Setting. We overcome the first scenario limitations by using our FDD implementation based simulator, which allowed us to run 3 separate scenarios, via which we tested the full functionalities of our FDD framework. In these scenarios we have successfully performed fault detection on the Light System, Liquid Discharge System, and Heating System

Chapter 7:

Conclusion and perspectives

Chapter 7.

Conclusion and perspectives

In this thesis we presented our contribution to Fault Detection and Diagnosis in the field of Ambient Intelligence. We start this chapter by a short recap of our contribution to Fault Detection and Diagnosis in Ambient Intelligent Systems, we give a summary of AmILoop, our FDD framework and the main issues treated by our approach, and then we give some perspectives for future works, finally we conclude.

7.1. Our Contribution

Our FDD framework presents a novel way to perform fault detection and diagnosis in Ambient Intelligent Systems.

The proposed approach is based on modeling the Ambient Environment and its heterogeneous components. The proposed modeling methodology allows the decoupling of actuators and sensors at design time, hence solving the problem of dynamicity in Ambient Intelligent Environment where devices of different types are added and removed at run-time.

The decoupling of sensors and actuators is made possible by the introduction of the concept of Effect. Effects allow the deduction of relationships between sensors and actuators at run-time, without requiring these relationships to be made explicit. An effect is a description of a physical phenomenon that takes place in the environment. To each effect is associated a hierarchy (from most detailed description to least detailed description) of Law-Sets. Law-Sets are sets of mathematical functions that calculate the values of some physical parameters based on known values of other physical parameters. It is through a series of calculations that the physical parameters produced by actuators and physical parameters detected by sensors are 'linked', thus deducing links between actuators and sensors at run-time, and at the same time predicting values that are supposed to be read by the sensors. The comparison of theoretical values and actual values of a certain sensor is the basis for the fault detection task.

At runtime, our FDD framework generates a multi-paradigm Prediction Model from a description of the structure of the Ambient System, its devices' behavior, and its Effects' Mathematical Models. The purpose of the Prediction Model is to allow the calculation of the sensors' expected values. Comparing the expected values with actual values finally enables the fault detection task.

We have integrated our Fault Detection Approach into a running Ambient Assisted Living application. And we have implemented our Fault Detection and Diagnosis framework AmILoop. We have used ModHel'X, a state of the art multi-paradigm model execution environment, as our execution platform. We have also developed a simulator in order to conduct experiments on various scenarios. The simulator uses the actual implementation; it just simulates the input and output of an ambient intelligent environment.

7.2. Perspectives

In this part we discuss some limitations discovered during this work, and we give some perspectives for future work in order to overcome these limitations.

7.2.1. Advanced Effect Modifier

As presented earlier the Modifier has a transformation law that defines how the effect changes when it go through the modifier. We considered this law as symmetric; however in a more realistic definition this law would be asymmetric. For instance, the glass of a window can be made with a one-way tint or film. In such a case we should have two laws defining the transformation law. We have not yet implemented this new approach, as the general case requires more work on the definition of the relation between the modifier and the zones it connects. Currently our abstract model represents such a relationship with:

```
Object hasProperty Property
```

which is finally instantiated with a *Modifier* and the *Zone*s it connects with:

```
door1 hasProperty kitchen
door1 hasProperty livingRoom
```

This definition describes two zones (kitchen and living room) connected via a door (door1). When a *Light Effect* crosses over from one zone to another we apply the same transformation law to calculate the amount of the effect that passed through.

However if the door was made of a one-way tinted glass we should define two transformation functions, moreover we should describe which side of the tinted glass is facing the kitchen and which side is facing the living room, which requires a different (more detailed) definition of the door and its modification laws, and of the relations between the door and the zones it connects.

Moreover we can have more than two zones connected with the same door, in which case an even more thorough definition is to be made as for what side of the tinted glass is facing which zone and what transformation law to use in the 6 cases (3 zones) of *Light Effect* passing through from zone to zone. The number of cases to consider will explode as the number of zones is higher.

7.2.2. Fault Diagnosis

As shown in our FDD architecture description in **Figure 47**, we do not impose a type of fault isolation techniques, nor a nature for the diagnosis model to be used. The latter's nature, however, can depend on the type of Diagnosis Engine used (for instance an ontology based model when we use a reasoning engine).

Usually the Fault Diagnosis task takes place after (sometimes simultaneously **[199]**) an inconsistency between the theoretical predicted value and the real value of a certain sensor has been detected (fault detection). A potential fault is to be located and diagnosed. After the fault

detection step we do have a list of involved devices and components that are in relation with the sensor reporting the failure. The fault diagnosis task aims to select, with a better precision, the component that is most likely the cause of the fault. We think that, once the fault detection task is performed (the Prediction Model is generated and the calculations have been evaluated), we dispose of an amount of information sufficient to apply different state of the art fault diagnosis techniques. Indeed, we know the relations between components, the existence or absence of other sensors connected to common actuators to which a sensor reporting a fault is also connected, the location of involved components and their tolerance values. In this thesis we have only detailed the fault detection part of the broader fault detection and diagnosis framework. However our FDD framework, as shown in **5.1.3**, could allow us to apply the idea of a probabilistic diagnosis approach. To explore this idea, in the next paragraph we define the diagnosis problem setup from a probabilistic point of view. We then cite some existing approaches for solving the probabilistic diagnosis problem. Finally we show some challenges that arise when applying these techniques in an Ambient Intelligent Environment.

7.2.2.1. **Using probabilistic approach for fault diagnosis**

Once a failure has been detected, ideally fault diagnosis would identify with 100% certainty the device causing the failure. However in reality this cannot be achieved. For instance even if two sensors receiving light from one light bulb returning both a failure, this does not mean that it is 100% certain that the bulb is faulty; in fact both sensors can fail simultaneously. Yet this case is very improbable, but we should be able to determine its probability. In reality, an FDD system tries to minimize the number of fault candidates in its diagnosis output, while ordering them from the most probable to the least probable. Ideally the diagnosis output is a single faulty component. However, since modeling errors can always occur, an FDD system that opts for the single faulty component solution has more risk of choosing the wrong component.

As we saw in **5.1.3**, when we have multiple sensors reading the same actuator, it is useful to compare the sensors' failure reports in order to decide which is "more probable" to be faulty: the actuator or one of the sensors. However the particular example we treated was relatively simple: two light sensors exposed to the light of a light bulb. If the first sensor reports a failure at the light bulb and the second sensor agrees, then the failure is "most probably" caused by the light bulb. However if the second sensor disagrees (meaning that it reads the value of Illuminance expected from the light bulb), then it is "most probably" the first sensor that is the cause of the detected failure. There are a lot of works that propose different probabilistic modeling, based on the Bayesian classification theory **[110]**, for resolving the diagnosis problem, such as **[200]**, **[201]**, **[202]**, and **[203]**.

A first issue with these probabilistic approaches is that the quality of their conclusions is based on a prior probability assumption about the value of **the probability that the sensor gives a correct reading even when it is broken**. This was demonstrated in **[204]**. The author introduces the **Transferable Belief Model** (TBM), which is a model that is based on an interpretation of the Dempster-Shafer approach **[205]** for representing quantified beliefs based on belief functions. The author applies the approach on different diagnosis problems (including finding the faulty sensor) **[206]** and concludes that, when our diagnosis problem lacks certain information, TBM gives more reliable diagnosis decision then classical Bayesian solutions.

In fact, there are already systems that use the probabilistic approach (or the TBM approach **[207]**) for deciding on what (only) sensors are most probably the faulty ones, using the readings of other sensors, when these sensors do not "agree". Such systems are used a lot in the industry for process control **[208]**, and in the field of aerospace engineering to avoid faults in the sensors, which in this case may lead to catastrophic system failure **[200]**. However, our motivation is to use the available sensor data in an Ambient Intelligent Environment in order to decide, when a

failure is detected, which component is responsible whether it is an actuator or a sensor. The problem is that actuators and sensors cannot be considered in a single probabilistic model, mainly because they have very different prior failure probabilities. Moreover they are of a very different nature when it comes to information production. A sensor reads information and sends it to the system, the information is processed and decisions are made accordingly. A probabilistic model to decide on the correctness of the information based on other information from other sensors is imaginable. The same thing cannot be done for actuators; hence we reckon that a different approach should be imagined that combines information from all system components, probably combining information from same types of components into separate probabilistic models. We would also need a novel way of drawing conclusions based on combining diagnosis results from each separate probabilistic model.

7.2.2.2. **Using Fault Trees for fault diagnosis**

Fault trees are an intuitive tool for displaying the binary relationships that will lead to failures. We reckon that they are more adapted to Ambient Assisted Living (or an Ambient Home setting in general), because of the relatively reduced number of components (compared to an Industrial plant for instance). They allow one to perform qualitative diagnosis, as opposed to the quantitative probabilistic diagnosis. Nevertheless quantitative failure probabilities deduced with the techniques discussed previously can be combined to be used with fault trees (see **Figure 44**).

We reckon that using fault trees is compatible with our Fault Detection and Diagnosis technique. In fact after the Fault Detection phase, we have a Prediction Model that contains the current connections between components. These connections could possibly be used to navigate in a Fault Diagnosis Tree through a series of questions in order to narrow down the possible causes of the fault, such as the fault tree described in **Figure 131**. The latter would constitute our Diagnosis Model in **Figure 47**.

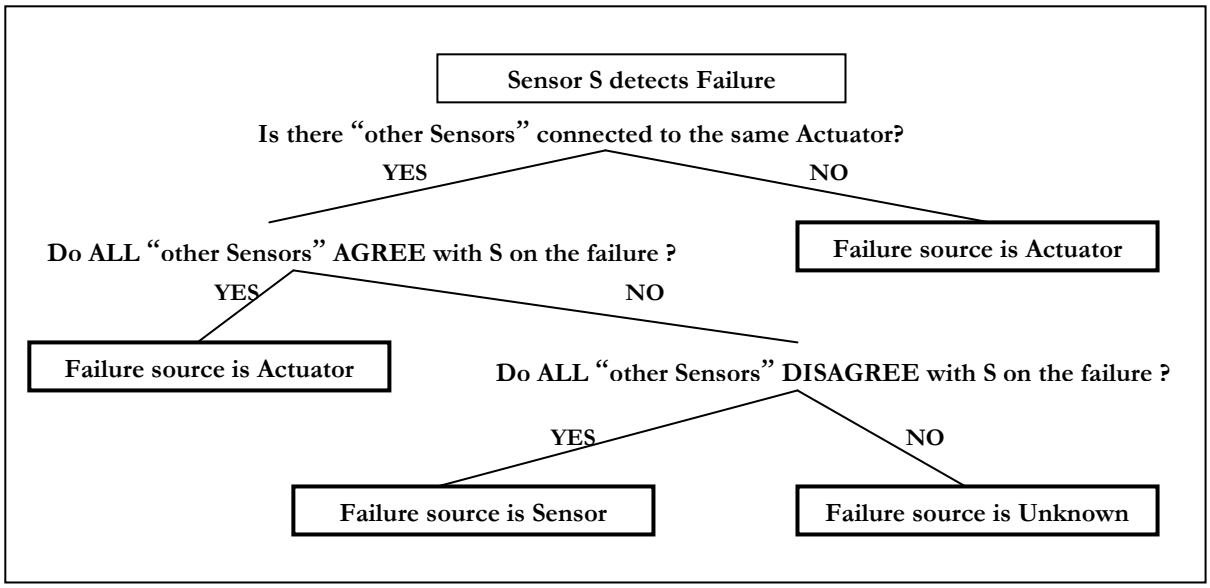

Figure 131. A simplified Decision tree for narrowing possible failure causes

The leaves here are the Diagnosis Conclusions. The "other Sensors" on the fault tree are other Sensors that are also connected with the same Actuator to which the Sensor S is connected. These "other sensors" can only deduced from the Prediction Model, hence after the fault detection task is done. This s consistent with our run-time architecture depicted in **Figure 51**

where the diagnosis engine uses the Diagnosis Model (the fault tree in this case) and the Prediction Model (extracting the "other Sensors" list in this case).

Note that this is a simplified Fault Tree. One would detail more branches handling more complicated cases (In particular for the *Failure source is Unknown* diagnosis conclusion) in order to have a better Fault Diagnosis conclusions. Also this fault tree handles only the case where a sensor is reporting a failure from one Actuator. A completely different fault tree should be defined for the case where the sensor reports a failure when it is monitoring the effects of many Actuators.

7.2.2.3. **Using Ontologies for diagnosis**

The reasoning presented in the previous paragraph can be expressed with rules that are used to reason on ontologies. Similar simple diagnosis rules (in the form of Horn clauses) were used for the CBDP AAL Application to perform diagnosis. However, in this case since we have one conclusion that we reach if all the premises are verified, every leaf in the previous decision tree would correspond to a conclusion, and the questions leading to it would constitute, among other conditions, the premises. A problem arises in this case when using Jena rules is that only the existence of instances can be matched in the premises. For instance we can not verify the non existence of other Sensors in order to conclude that the Actuator is Faulty after answering the first question in the previous decision tree. So the rule that decides that *Failure source in Unknown* can be expressed with the following Jena rule:

```
[ DIAGNOSIS_SENSOR:

    (?DLS RDF:type AMI:LightSensor),
    (?OLS1 RDF:type AMI:LightSensor),
    (?OLS2 RDF:type AMI:LightSensor),
    (?LA RDF:type AMI:LightActuator),
    (?R RDF:type ?RT),
    (?RT RDFS:subClassOf AMI:Room),
    (?DLS AMI:isIn ?R),
    (?OLS1 AMI:isIn ?R),
    (?OLS2 AMI:isIn ?R),
    (?LA AMI:isIn ?R),
    (?DLS AMI:faultDetection 'failure'),
    (?OLS1 AMI:faultDetection 'failure'),
    (?OLS2 AMI:faultDetection 'sucess')

            -> (?DLS AMI:faultDiagnosis 'unknown') ]
```

In other words: **If there is a least one other Sensor** (here we answered the other sensors existence first question) **that detects a failure, and another Sensor that does not detect a failure** we can answer the question: "*Do ALL other Sensors DISAGREE with S on the failure?"* with **No**, thus concluding that the Source of the Fault is **unknown**.

In the AAL application we used the specific ALL application bundle to launch simple and special cases rules (for example: 1 sensor 2 actuators) corresponding to specific predetermined scenarios. We reckon that we can create more generic diagnosis rules that verify the non-existence of instances, in order to perform diagnosis, with other RDF query languages such as SPARQL **[209]**, which has the EXIST/NOT EXIST built-in functions.

On a more general scope, ontologies can represent sophisticated knowledge about a specific domain and can describe in great detail contextual information about a system, hence allowing to reason over this knowledge in order to deduce precise conclusions. This can be used for fault

diagnosis. For instance work in **[210]** uses ontologies to represent different aspects of an electrical network from a diagnosis point of view, and uses these ontologies to perform Fault Diagnosis on the electrical network. We reckon that an **Ontology** can be used by our FDD framework as the **Diagnosis Model** and an **Ontology Reasoning Engine** can be sued as **Diagnosis Engine**, in order to perform accurate Fault Diagnosis. The diagnosis ontology would be constantly updated with contextual information from the Prediction Model such as detected faults, predicted and actual sensor values, links between components, etc. The Diagnosis engine would deduce Diagnosis reports based on these information and other domain specific information that should also be represented in the diagnosis ontology.

7.2.3. Considering the user

In the context of our Fault Detection and Diagnosis framework, it would be an interesting addition to consider the user's feedback, in order to confirm, or refute a diagnosis report. If stored by the Ambient Intelligent System, this information can even be useful for future diagnosis. In the CBDP Ambient Assisted Living Application, the framework stores users' feedbacks in the ontology. However this information is not actually used for future diagnosis, since the rules that reason over the stored feedbacks are to be defined.

Note that our FDD framework only supervises the Ambient System's actions (done via actuators) in order to detect malfunctions and does not supervise the users' actions. In fact, unlike system actions, users' behaviors are unpredictable and require different modeling and fault diagnosis approach. The field of users tasks supervision and assistance in Ambient Intelligent Systems is a new field. Research in this field aims at helping users to perform their tasks properly, and detecting any unwanted or dangerous deviations of normal users' behaviors, in order to help them complete their wanted tasks. Some works have been done in that context that aim to creating a novel ambient user task model, which takes into consideration the specificities of user tasks in an Ambient Intelligent Environment setting **[211]**.

7.3. Conclusion

Ambient Intelligent Systems are a fast growing trend in modern societies that is enhancing people's way of life. More and more people rely on these systems for their everyday activities without even noticing them. This makes reliability of such systems a very important property, and the self-diagnosis capability of such systems a very critical characteristic.

Our work falls within this general idea of equipping Ambient Intelligent systems with auto diagnosis capabilities. With our proposed approach we have overcome particular challenges that arise in Fault Detection and Diagnosis in the Ambient Intelligence context. We have also proposed directions for future works in order to complete our research in Fault Detection and Diagnosis in Ambient Intelligence.

Annexes

Annex-A

A. Light as a physical phenomenon – definitions [179][180]

A.1 Radiometry

Radiometry is the science of the measurement of electromagnetic radiation. The basic concepts that describe the power and distribution of the radiation are:

- **Radiant Flux**

Radiant flux is the total radiant power emitted from a light source. In some cases, like when we are considering light that is passing through a certain area, radiant power can be defined as the rate of flow of radiant energy passing through that area. The SI unit of radiant flux is the *Watt*.

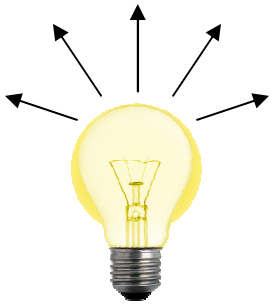

Figure 132. Radiant flux

- **Radiant Intensity**

Radiant intensity is the directed density of radiation from a source in accordance to a solid angle (defines the size of the cone of intensity radiated by a source point), noted Ω. The value of a radiant intensity emitted by a light source in a given direction is the sum of the power of all the rays that are following that direction.

The SI unit for radiant intensity is Watt per steradian, noted W/sr.

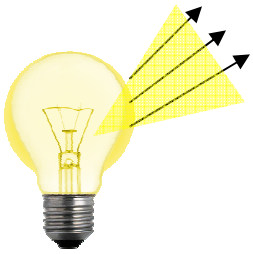

Figure 133. Radiant intensity

- **Irradiance**

Irradiance is the measure of radiant flux incidence on an object's surface.

The SI unit for irradiance is Watt per square meter, noted W/m^2.

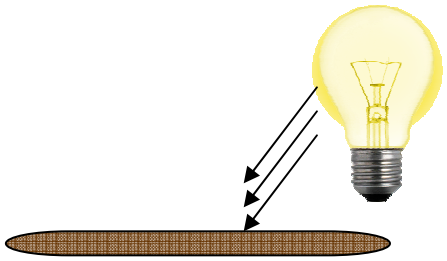

Figure 134. Irradiance

- **Radiance**

Radiance is the quantity of radiation calculated when the emitted light beam falls on a surface following a solid angle Ω and in a specified direction.

The SI unit for radiance is Watt per square meter steradian, noted $Watt/m^2.sr$.

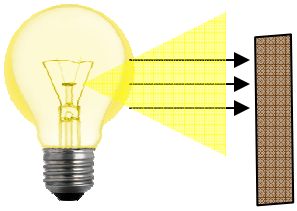

Figure 135. Radiance

A.2 Spectroradiometry

The field of Spectroradiometry aims to measure the energy of light within the electromagnetic spectrum. The energy can be measured over the entire spectrum of wavelengths or within a specific band of wavelengths.

- **Spectral Irradiance**

Spectral Irradiance is the measure of the total radiant flux per surface unit projected on a surface at a particular wavelength. The wavelengths of energy values are measured in nanometer (nm).

The SI unit for spectral radiance is Watt per square meter nanometer, noted $W/m^2.nm$.

- **Spectral Radiance**

The spectral radiance is the sum of all energy measured over a spectrum. The radiations fall on the surface in a specific direction and following a solid angle Ω. The SI unit for spectral radiance is Watt per square meter steradian nanometer, noted $W/m^2.sr.nm$.

A.3 Photometry

Photometry focuses on measuring the characteristics of the electromagnetic energy that are visible to the human eye, hence the use of the term 'luminous', which refers to visible light.

The basic concepts that make photometry are:

- **Luminous flux (what light sources produce)**

Analogically to the radiant flux, which is the total radiant power emitted from a light source, luminous flux is the flow of light energy emitted by a source that is visible by the human eye. The

value is calculated from the radiant flux by applying the luminosity function V(λ), which is a standard function established by the CIE[1] to convert radiant energy into luminous energy visible by the 'standard eye'.

The V(λ) function defines how the average human eye is sensitive to different wavelengths (interpreted by the human eye with different light colors) of light.

The SI unit for luminous flux is lumen, noted *lm*.

Light bulbs' powers are described in Watt. To convert from Watt to lumen we need to know the value of the luminous efficacy of the light source η_v. The luminous efficacy is the ratio of luminous flux (*lm*) emitted by the source to the input power (*watt*) **[178]**. To calculate the luminous efficacy we need to determine two entities: the radiant efficiency of the source (ratio of output radiant flux to input power), noted η_e, and the luminous efficacy of radiation (ratio of luminous flux to radiant flux), noted K. η_v is then calculated by:

$\eta_v = \eta_e . K$

Using this we can draw an approximate luminous efficacy table for different known type of commercial light sources (see **Table 3 [213]**)

Light type	Approximate luminous efficacy (lumens/watt)
incandescent light bulb	12-17 lm/W
Halogen lamp	16-24 lm/W
Fluorescent lamp	45-75 lm/W
MCOB LED lamp	100-130 lm/W
Metal halide lamp	75-100 lm/W
High pressure sodium vapor lamp	85-150 lm/W
Low pressure sodium vapor lamp	100-200 lm/W
Mercury vapor lamp	35-65 lm/W

Table 3. Luminous efficacy table for known lamp types

For example the luminous flux of a 30 Watts Fluorescent lamp that has luminous efficacy of 50 lumens per Watt is:

30 W x 50 lm/W = 1500 lm

- **Luminous intensity**

This expresses the power of a light source. It is defined as the quantity of luminous flux emitted in a given direction per solid angle Ω (per steradian).

The unit is candela, noted *cd*.

To calculate the luminous intensity in candela knowing the luminous flux in lumens of the source, we divide the luminous flux value by the solid angle Ω in steradians:

$I_v = \Phi_v / \Omega$

To deduce the value of the solid angle Ω in steradians we use:

[1] CIE stands for 'Commission Internationale de l' Éclairage' or 'International Commission on Illumination'. It is the international authority on light, illumination, color, and color spaces.

$\Omega = 2\pi[1 - \cos(\theta/2)]$

Where θ is the angle of the light beam emitted from the light source to a certain surface.

- **Illuminance (what light sensors detect)**

Illuminance is luminous flux per area unit or at a point.

The SI unit for Illuminance is lux, noted *lx*. Where 1 $lx = 1\ lm/m^2$.

The next table depicts some examples of Illuminance provided under natural conditions **[214]**:

:**Illuminance**	**Surfaces illuminated by:**
10−4 lux	Moonless, overcast night sky (starlight)
0.002 lux	Moonless clear night sky with airglow
0.27–1.0 lux	Full moon on a clear night
400 lux	Sunrise or sunset on a clear day.
10,000–25,000 lux	Full daylight (not direct sun)
32,000–130,000 lux	Direct sunlight

Table 4. Examples of Illuminance values under natural conditions

- **Luminance**

Luminance is also called brightness in photometry. It is the luminous intensity emitted in a certain direction from a light source per unit area.

The SI unit is the candelas per square meter, noted cd/m^2.

Annex-B

Table of specific heat capacities [181][215]

Table of specific heat capacities at 25 °C (298 K)	Phase	(mass) specific heat capacity c_p or c_m $J \cdot g^{-1} \cdot K^{-1}$	Constant pressure molar heat capacity $C_{p,m}$ $J \cdot mol^{-1} \cdot K^{-1}$	Constant volume molar heat capacity $C_{v,m}$ $J \cdot mol^{-1} \cdot K^{-1}$	Volumetric heat capacity C_v $J \cdot cm^{-3} \cdot K^{-1}$	Constant vol. atom-molar heat capacity in units of R $C_{v,m(atom)}$ atom-mol^{-1}
Air (Sea level, dry, 0 °C (273.15 K))	gas	1.0035	29.07	20.7643	0.001297	~ 1.25 R
Air (typical room conditions)	gas	1.012	29.19	20.85	0.00121	~ 1.25 R
Aluminum	solid	0.897	24.2		2.422	2.91 R
Antimony	solid	0.207	25.2		1.386	3.03 R
Argon	gas	0.5203	20.7862	12.4717		1.50 R
Arsenic	solid	0.328	24.6		1.878	2.96 R
Beryllium	solid	1.82	16.4		3.367	1.97 R
Cadmium	solid	0.231	26.02			3.13 R
Carbon dioxide CO_2	gas	0.839*	36.94	28.46		1.14 R
Chromium	solid	0.449	23.35			2.81 R
Copper	solid	0.385	24.47		3.45	2.94 R
Ethanol	liquid	2.44	112		1.925	1.50 R
Gasoline (octane)	liquid	2.22	228		1.64	1.05 R
Gold	solid	0.129	25.42		2.492	3.05 R
Granite	solid	0.790			2.17	
Graphite	solid	0.710	8.53		1.534	1.03 R
Helium	gas	5.1932	20.7862	12.4717		1.50 R
Hydrogen	gas	14.30	28.82			1.23 R
Iron	solid	0.450			3.537	3.02 R
Lead	solid	0.129	26.4		1.44	3.18 R
Lithium	solid	3.58	24.8		1.912	2.98 R
Magnesium	solid	1.02	24.9		1.773	2.99 R
Mercury	liquid	0.1395	27.98		1.888	3.36 R
Nitrogen	gas	1.040	29.12	20.8		1.25 R
Neon	gas	1.0301	20.7862	12.4717		1.50 R
Oxygen	gas	0.918	29.38	21.0		1.26 R

Table of specific heat capacities at 25 °C (298 K)	**Phase**	**(mass) specific heat capacity c_p or c_m $J \cdot g^{-1} \cdot K^{-1}$**	**Constant pressure molar heat capacity $C_{p,m}$ $J \cdot mol^{-1} \cdot K^{-1}$**	**Constant volume molar heat capacity $C_{v,m}$ $J \cdot mol^{-1} \cdot K^{-1}$**	**Volumetric heat capacity C_v $J \cdot cm^{-3} \cdot K^{-1}$**	**Constant vol. atom-molar heat capacity in units of R $C_{v,m(atom)}$ atom-mol^{-1}**
Silver	solid	0.233	24.9		2.44	2.99 R
Sodium	solid	1.230	28.23			3.39 R
Steel	solid	0.466				
Tin	solid	0.227	27.112			3.26 R
Titanium	solid	0.523	26.060			3.13 R
Tungsten	solid	0.134	24.8		2.58	2.98 R
Uranium	solid	0.116	27.7		2.216	3.33 R
Water at 100 °C (steam)	gas	2.080	37.47	28.03		1.12 R
Water at 25 °C	liquid	4.1813	75.327	74.53	4.1796	3.02 R
Water at 100 °C	liquid	4.1813	75.327	74.53	4.2160	3.02 R
Water at −10 °C (ice)	solid	2.11	38.09		1.938	1.53 R
Zinc	solid	0.387	25.2		2.76	3.03 R

Table 5. Table of specific heat capacities

Annex–C

Test 1:

Evaluating the Mathematical Model and Validation for results reported in **Figure 117**:

sensor_1 : According the trace in the figure, the value calculated by **L4** for **sensor_1** should be **812.5**. The 2D Light Law Set calls are as following:

Call to **L4**:

```
ambientLightIntensity(sensor_1)=
directLightExposure(sensor_1,bulb_1)+
directLightExposure(sensor_1,bulb_2)+
directLightExposure(sensor_1,bulb_3)+
directLightExposure(sensor_1,doubleHingedDoor)+
directLightExposure(sensor_1,slidingDoor)+
directLightExposure(sensor_1,windowBlinds)
```

- this generates the following calls to the Law **L3**; one for each device

```
directLightExposure(sensor_1,bulb_1)=
luminousFlux(bulb_1) / distance(sensor_1,bulb_1)^2

directLightExposure(sensor_1,bulb_2)=
luminousFlux(bulb_2) / distance(sensor_1,bulb_2)^2

directLightExposure(sensor_1,bulb_3)=
luminousFlux(bulb_3) / distance(sensor_1,bulb_3)^2

directLightExposure(sensor_1,doubleHingedDoor)=
luminousFlux(doubleHingedDoor) / distance(sensor_1,doubleHingedDoor)^2

directLightExposure(sensor_1,slidingDoor)=
luminousFlux(slidingDoor) / distance(sensor_1,slidingDoor)^2

directLightExposure(sensor_1,windoBlinds)=
luminousFlux(windoBlinds) / distance(sensor_1,windoBlinds)^2
```

with

luminousFlux(bulb_1)=1750

luminousFlux(bulb_2)=50 (first second of heating phase)

luminousFlux(bulb_3)=800

luminousFlux(doubleHingedDoor)=0*ambientLightIntensity(doubleHingedDoor)=0

because **doubleHingedDoor.AmbientLightModificationRatio=0** (door closed).

Same for other closed Effect Modifiers:

luminousFlux(slidingDoor)=0
luminusFlux(windoBlinds)=0

- this generates the following calls to the Law **L5**; one for each component
- before evaluating **L5**, each **L5** generates the following calls to **L1** (the same-zone verification function)

sameZone(sensor_1,bulb_1) returns **FALSE (room2!=room1).**

sameZone(sensor_1,bulb_2) returns **TRUE (room2=room2).**

sameZone(sensor_1,bulb_3) returns **TRUE (room2=room2).**

sameZone(sensor_1,doubleHingedDoor) returns **TRUE (room2=room2).**

sameZone(sensor_1,slidingDoor) returns **FALSE (room2!=room1).**

sameZone(sensor_1,windowBlinds) returns **FALSE (room2!=room1).**

So now the calls to **L5** look like the following:

distance(sensor_1,bulb_1)= ∞

distance(sensor_1,bulb_2)=
Sqrt[(x(sensor_1)-x(bulb_2))^2 + (y(sensor_1)-y(bulb_2))^2]

distance(sensor_1,bulb_3)=
Sqrt[(x(sensor_1)-x(bulb_3))^2 + (y(sensor_1)-y(bulb_3))^2]

distance(sensor_1,doubleHingedDoor)=
Sqrt[(x(sensor_1)-x(doubleHingedDoor))^2 +
(y(sensor_1)-y(doubleHingedDoor))^2]

distance(sensor_1,slidingDoor)= ∞

distance(sensor_1,windowBlinds)= ∞

with

x(sensor_1)=3 y(sensor_1)=4
x(bulb_2)=3 y(bulb_2)=2
x(bulb_3)=4 y(bulb_3)=4
x(doubleHingedDoor)=2 y(doubleHingedDoor)=2
x(slidingDoor)=1 y(slidingDoor)=0
x(windowBlinds)=0 y(windowBlinds)=3

which give

distance(sensor_1,bulb_1)= ∞
distance(sensor_1,bulb_2)=2
distance(sensor_1,bulb_3)=1
distance(sensor_1,doubleHingedDoor)=Sqrt(5)
distance(sensor_1,slidingDoor)= ∞
distance(sensor_1,windowBlinds)= ∞

using these values to evaluate previous **L3** calls gives

directLightExposure(sensor_1,bulb_1)=1750/∞
directLightExposure(sensor_1,bulb_2)=50/4
directLightExposure(sensor_1,bulb_3)=800/1
directLightExposure(sensor_1,doubleHingedDoor)=0/5
directLightExposure(sensor_1,slidingDoor)=0/∞
directLightExposure(sensor_1,windowBlinds)=0/∞

Finally the call to L4 for sensor_1 gives

ambientLightIntensity(sensor_1)=0+12.5+800+0+0+0= 812.5

That is equal to the value predicted by the Prediction Model.

sensor_2 : The value calculated by **L4** for **sensor_2** should be **437.5** The 2D Light Law Set calls are as following:

Call to **L4**:

```
ambientLightIntensity(sensor_2)=
directLightExposure(sensor_2,bulb_1)+
directLightExposure(sensor_2,bulb_2)+
directLightExposure(sensor_2,bulb_3)+
directLightExposure(sensor_2,doubleHingedDoor)+
directLightExposure(sensor_2,slidingDoor)+
directLightExposure(sensor_2,windowBlinds)
```

- this generates the following calls to the Law **L3**; one for each device

```
directLightExposure(sensor_2,bulb_1)=
luminousFlux(bulb_1) / distance(sensor_2,bulb_1)^2

directLightExposure(sensor_2,bulb_2)=
luminousFlux(bulb_2) / distance(sensor_2,bulb_2)^2

directLightExposure(sensor_2,bulb_3)=
luminousFlux(bulb_3) / distance(sensor_2,bulb_3)^2

directLightExposure(sensor_2,doubleHingedDoor)=
```

luminousFlux(doubleHingedDoor) / distance(sensor_2,doubleHingedDoor)^2

directLightExposure(sensor_2,slidingDoor)=
luminousFlux(slidingDoor) / distance(sensor_2,slidingDoor)^2

directLightExposure(sensor_2,windoBlinds)=
luminousFlux(windoBlinds) / distance(sensor_2,windoBlinds)^2

with

luminousFlux(bulb_1)=1750

luminousFlux(bulb_2)=50 (first second of heating phase)

luminousFlux(bulb_3)=800

luminousFlux(doubleHingedDoor)=0*ambientLightIntensity(doubleHingedDoor)=0

because **doubleHingedDoor.AmbientLightModificationRatio=0** (door closed).

Same for other closed Effect Modifiers:

luminousFlux(slidingDoor)=0

luminusFlux(windoBlinds)=0

- this generates the following calls to the Law **L5**; one for each component
- before evaluating **L5**, each **L5** generates the following calls to **L1** (the same-zone verification function)

sameZone(sensor_2,bulb_1) returns **TRUE (room1=room1).**

sameZone(sensor_2,bulb_2) returns **FALSE (room1!=room2).**

sameZone(sensor_2,bulb_3) returns **FALSE (room1!=room2).**

sameZone(sensor_2,doubleHingedDoor) returns **FALSE (room1!=room2).**

sameZone(sensor_2,slidingDoor) returns **TRUE (room1=room1).**

sameZone(sensor_2,windowBlinds) returns **TRUE (room1=room1).**

So now the calls to **L5** look like the following:

distance(sensor_2,bulb_1)=
Sqrt[(x(sensor_2)-x(bulb_1))^2 + (y(sensor_2)-y(bulb_1))^2]

distance(sensor_2,bulb_2)= ∞

distance(sensor_2,bulb_3)= ∞

distance(sensor_2,doubleHingedDoor)=
Sqrt[(x(sensor_2)-x(doubleHingedDoor))^2 +
(y(sensor_2)-y(doubleHingedDoor))^2]

```
distance(sensor_2,slidingDoor)=
Sqrt[(x(sensor_2)-x(slidingDoor))^2 +
(y(sensor_2)-y(slidingDoor))^2]

distance(sensor_2,windowBlinds)=
Sqrt[(x(sensor_2)-x(windowBlinds))^2 +
(y(sensor_2)-y(windowBlinds))^2]
```

with

```
x(sensor_2)=1 y(sensor_2)=4
x(bulb_1)=1 y(bulb_1)=2
x(doubleHingedDoor)=2 y(doubleHingedDoor)=2
x(slidingDoor)=1 y(slidingDoor)=0
x(windowBlinds)=0 y(windowBlinds)=3
```

which give

```
distance(sensor_2,bulb_1)=2
distance(sensor_2,bulb_2)= ∞
distance(sensor_2,bulb_3)= ∞
distance(sensor_1,doubleHingedDoor)=Sqrt(5)
distance(sensor_2,slidingDoor)=4
distance(sensor_2,windowBlinds)=Sqrt(2)
```

using these values to evaluate previous **L3** calls gives

```
directLightExposure(sensor_2,bulb_1)=1750/4
directLightExposure(sensor_2,bulb_2)=50/∞
directLightExposure(sensor_2,bulb_3)=800/∞
directLightExposure(sensor_2,doubleHingedDoor)=0/5
directLightExposure(sensor_2,slidingDoor)=0/4
directLightExposure(sensor_2,windowBlinds)=0/2
```

Finally the call to L4 for sensor_1 gives

```
ambientLightIntensity(sensor_1)=437.5+0+0+0+0+0= 437.5
```

That is equal to the value predicted by the Prediction Model.

sensor_3 : Reports the Illuminance of the outside without: 100000 lm. No calculations are performed as it is defined as a special sensor (trusted sensor of an Effect Modifier).

Evaluating the Mathematical Model and Validation n for results reported in **Figure 118**:

The only value that changed after 30 seconds is the value of sensor_1, it is because it is exposed to the Fluorescent Light Bulb in the room, which reached its maximum capacity after 30 seconds:

sensor_1 : According the trace in the figure, the value calculated by **L4** for **sensor_1** should be **1175**. The 2D Light Law Set calls are as following:

Call to **L4**:

```
ambientLightIntensity(sensor_1)=
directLightExposure(sensor_1,bulb_1)+
directLightExposure(sensor_1,bulb_2)+
directLightExposure(sensor_1,bulb_3)+
directLightExposure(sensor_1,doubleHingedDoor)+
directLightExposure(sensor_1,slidingDoor)+
directLightExposure(sensor_1,windowBlinds)
```

- this generates the following calls to the Law **L3**; one for each device

```
directLightExposure(sensor_1,bulb_1)=
luminousFlux(bulb_1) / distance(sensor_1,bulb_1)^2

directLightExposure(sensor_1,bulb_2)=
luminousFlux(bulb_2) / distance(sensor_1,bulb_2)^2

directLightExposure(sensor_1,bulb_3)=
luminousFlux(bulb_3) / distance(sensor_1,bulb_3)^2

directLightExposure(sensor_1,doubleHingedDoor)=
luminousFlux(doubleHingedDoor) / distance(sensor_1,doubleHingedDoor)^2

directLightExposure(sensor_1,slidingDoor)=
luminousFlux(slidingDoor) / distance(sensor_1,slidingDoor)^2

directLightExposure(sensor_1,windoBlinds)=
luminousFlux(windoBlinds) / distance(sensor_1,windoBlinds)^2
```

with

`luminousFlux(bulb_1)=1750`

`luminousFlux(bulb_2)=1500` (CFL reached its maximum capacity)

`luminousFlux(bulb_3)=800`

`luminousFlux(doubleHingedDoor)=0*TotalIllumination2D(doubleHingedDoor)=0`

because `doubleHingedDoor.AmbientLightModificationRatio=0` (door closed).

Same for other closed Modifiers:

`luminousFlux(slidingDoor)=0`

`luminusFlux(windoBlinds)=0`

- this generates the following calls to the Law **L5**; one for each component

- before evaluating **L5**, each **L5** generates the following calls to **L1** (the same-zone verification function)

sameZone(sensor_1,bulb_1) returns **FALSE (room2!=room1).**

sameZone(sensor_1,bulb_2) returns **TRUE (room2=room2).**

sameZone(sensor_1,bulb_3) returns **TRUE (room2=room2).**

sameZone(sensor_1,doubleHingedDoor) returns **TRUE (room2=room2).**

sameZone(sensor_1,slidingDoor) returns **FALSE (room2!=room1).**

sameZone(sensor_1,windowBlinds) returns **FALSE (room2!=room1).**

So now the calls to **L5** look like the following:

distance(sensor_1,bulb_1)= ∞

distance(sensor_1,bulb_2)=
Sqrt[(x(sensor_1)-x(bulb_2))^2 + (y(sensor_1)-y(bulb_2))^2]

distance(sensor_1,bulb_3)=
Sqrt[(x(sensor_1)-x(bulb_3))^2 + (y(sensor_1)-y(bulb_3))^2]

distance(sensor_1,doubleHingedDoor)=
Sqrt[(x(sensor_1)-x(doubleHingedDoor))^2 +
(y(sensor_1)-y(doubleHingedDoor))^2]

distance(sensor_1,slidingDoor)= ∞

distance(sensor_1,windowBlinds)= ∞

with

x(sensor_1)=3 y(sensor_1)=4
x(bulb_2)=3 y(bulb_2)=2
x(bulb_3)=4 y(bulb_3)=4
x(doubleHingedDoor)=2 y(doubleHingedDoor)=2
x(slidingDoor)=1 y(slidingDoor)=0
x(windowBlinds)=0 y(windowBlinds)=3

which give

distance(sensor_1,bulb_1)= ∞
distance(sensor_1,bulb_2)=2
distance(sensor_1,bulb_3)=1

distance(sensor_1,doubleHingedDoor)=Sqrt(5)

distance(sensor_1,slidingDoor)= ∞

distance(sensor_1,windowBlinds)= ∞

using these values to evaluate previous **L3** calls gives

directLightExposure(sensor_1,bulb_1)=1750/∞

directLightExposure(sensor_1,bulb_2)=1500/4

directLightExposure(sensor_1,bulb_3)=800/1

directLightExposure(sensor_1,doubleHingedDoor)=0/5

directLightExposure(sensor_1,slidingDoor)=0/∞

directLightExposure(sensor_1,windowBlinds)=0/∞

Finally the call to L4 for sensor_1 gives

ambientLightIntensity(sensor_1)=0+375+800+0+0+0= 1175

That is equal to the value predicted by the Prediction Model.

sensor_2 : Does not change value (**437.5** lux), since it is not exposed to a CFL bulb.

Test 2:

Evaluating the Mathematical Model and Validation for results reported in **Figure 119**:

sensor_1 : According the trace in the figure, the value calculated by **L4** for **sensor_1** should be **1525**. The 2D Light Law Set calls are as following:

Call to **L4**:

```
ambientLightIntensity(sensor_1)=
directLightExposure(sensor_1,bulb_1)+
directLightExposure(sensor_1,bulb_2)+
directLightExposure(sensor_1,bulb_3)+
directLightExposure(sensor_1,doubleHingedDoor)+
directLightExposure(sensor_1,slidingDoor)+
directLightExposure(sensor_1,windowBlinds)
```

- this generates the following calls to the Law **L3**; one for each device

```
directLightExposure(sensor_1,bulb_1)=
luminousFlux(bulb_1) / distance(sensor_1,bulb_1)^2

directLightExposure(sensor_1,bulb_2)=
luminousFlux(bulb_2) / distance(sensor_1,bulb_2)^2

directLightExposure(sensor_1,bulb_3)=
luminousFlux(bulb_3) / distance(sensor_1,bulb_3)^2
```

directLightExposure(sensor_1,doubleHingedDoor)=
luminousFlux(doubleHingedDoor) / distance(sensor_1,doubleHingedDoor)^2

directLightExposure(sensor_1,slidingDoor)=
luminousFlux(slidingDoor) / distance(sensor_1,slidingDoor)^2

directLightExposure(sensor_1,windoBlinds)=
luminousFlux(windoBlinds) / distance(sensor_1,windoBlinds)^2

with

luminousFlux(bulb_1)=1750

luminousFlux(bulb_2)=1500 (CFL reached its maximum capacity)

luminousFlux(bulb_3)=800

luminousFlux(doubleHingedDoor)=1*ambientLightIntensity(doubleHingedDoor)=0

because **doubleHingedDoor.AmbientLightModificationRatio=1** (door opened).

Other Modifiers are kept closed:

luminousFlux(slidingDoor)=0

luminusFlux(windoBlinds)=0

Now that the double hinged door is fully opened, we have to evaluate

ambientLightIntensity(doubleHingedDoor)in room1 to estimate how much the doubleHingedDoor contributes in luminous flux in room2

So:

luminousFlux(doubleHingedDoor)=1*ambientLightIntensity(doubleHingedDoor)
in room2 in room1

The call is treated as if it was a call from a sensor, but with ignoring the effect modifier doubleHingedDoor in the calculations.

So, unlike when sensor_2 called **ambientLightIntensity** in room1, when the doubleHingedDoor calls it, it generates 5 calls to directLightExposure instead of 6 calls. These calls are highlighted in blue:

doubleHingedDoor :

Call to **L4**:

ambientLightIntensity(doubleHingedDoor)=
directLightExposure(doubleHingedDoor,bulb_1)+
directLightExposure(doubleHingedDoor,bulb_2)+
directLightExposure(doubleHingedDoor,bulb_3)+
directLightExposure(doubleHingedDoor,doubleHingedDoor)+
directLightExposure(doubleHingedDoor,slidingDoor)+
directLightExposure(doubleHingedDoor,windowBlinds)

- this generates the following calls to the Law **L3**; one for each device

directLightExposure(doubleHingedDoor,bulb_1)=
luminousFlux(bulb_1) / distance(doubleHingedDoor,bulb_1)^2

directLightExposure(doubleHingedDoor,bulb_2)=
luminousFlux(bulb_2) / distance(doubleHingedDoor,bulb_2)^2

directLightExposure(doubleHingedDoor,bulb_3)=
luminousFlux(bulb_3) / distance(doubleHingedDoor,bulb_3)^2

directLightExposure(doubleHingedDoor,doubleHingedDoor)=0

directLightExposure(doubleHingedDoor,slidingDoor)=
luminousFlux(slidingDoor) / distance(doubleHingedDoor,slidingDoor)^2

directLightExposure(doubleHingedDoor,windoBlinds)=
luminousFlux(windoBlinds) / distance(doubleHingedDoor,windoBlinds)^2

with

luminousFlux(bulb_1)=1750
luminousFlux(bulb_2)=1500
luminousFlux(bulb_3)=800
luminousFlux(slidingDoor)=0
luminusFlux(windoBlinds)=0

(Closed Effect Modifiers):

- this generates the following calls to the Law **L5**; one for each component
- before evaluating **L5**, each **L5** generates the following calls to **L1** (the same-zone verification function)

doubleHingedDoor has room1 and room2 as Zones in property, but in these calculations it is considered as part of room1 (we want to calculate how much of light from room1 is going to room2).

sameZone returns FALSE when comparing zones of same objects for additional safety (so the Effect Modifier doesn't consider itself in the calculations).

sameZone(doubleHingedDoor,bulb_1) returns **TRUE (room1=room1).**

sameZone(doubleHingedDoor,bulb_2) returns **FALSE (room1!=room2).**

sameZone(doubleHingedDoor,bulb_3) returns **FALSE (room1!=room2).**

sameZone(doubleHingedDoor,doubleHingedDoor) returns **FALSE.**

sameZone(doubleHingedDoor,slidingDoor) returns **TRUE (room1=room1).**

sameZone(doubleHingedDoor,windowBlinds) returns **TRUE (room1=room1).**

So now the calls to **L5** look like the following:

```
distance(doubleHingedDoor,bulb_1)=
Sqrt[(x(doubleHingedDoor)-x(bulb_1))^2+
(y(doubleHingedDoor)-y(bulb_1))^2]

distance(doubleHingedDoor,bulb_2)= ∞

distance(doubleHingedDoor,bulb_3)= ∞

distance(doubleHingedDoor,doubleHingedDoor)= ∞ (also for additional safety)

distance(doubleHingedDoor,slidingDoor)=
Sqrt[(x(doubleHingedDoor)-x(slidingDoor))^2 +
(y(doubleHingedDoor)-y(slidingDoor))^2]

distance(doubleHingedDoor,windowBlinds)=
Sqrt[(x(doubleHingedDoor)-x(windowBlinds))^2 +
(y(doubleHingedDoor)-y(windowBlinds))^2]
```

with

```
x(doubleHingedDoor)=2 y(doubleHingedDoor)=2
          x(bulb_1)=1 y(bulb_1)=2
     x(slidingDoor)=1 y(slidingDoor)=0
    x(windowBlinds)=0 y(windowBlinds)=3
```

which give

```
distance(doubleHingedDoor,bulb_1)=1
distance(doubleHingedDoor,bulb_2)= ∞
distance(doubleHingedDoor,bulb_3)= ∞
distance(doubleHingedDoor,doubleHingedDoor)= ∞
distance(doubleHingedDoor,slidingDoor)=Sqrt(3)
distance(doubleHingedDoor,windowBlinds)=Sqrt(3)
```

using these values to evaluate previous **L3** calls gives

```
directLightExposure(doubleHingedDoor,bulb_1)=1750/1
directLightExposure(doubleHingedDoor,bulb_2)=1500/∞
directLightExposure(doubleHingedDoor,bulb_3)=800/∞
directLightExposure(doubleHingedDoor,doubleHingedDoor)=0
directLightExposure(doubleHingedDoor,slidingDoor)=0/3
directLightExposure(doubleHingedDoor,windowBlinds)=0/3
```

Finally the call to L4 for sensor_1 gives

ambientLightIntensity(doubleHingedDoor)=1750+0+0+0+0+0= 1750

So **luminousFlux(doubleHingedDoor)=1*1750=1750**

Now that we know the value of **luminousFlux(doubleHingedDoor)=1750**, we continue our calculations as if the doubleHingedDoor was another actuator in room2.

We return to Room2.

We left at estimating **directLightExposure** for every component.

- this generates the following calls to the Law **L5**; one for each component
- before evaluating **L5**, each **L5** generates the following calls to **L1** (the same-zone verification function)

sameZone(sensor_1,bulb_1) returns **FALSE (room2!=room1).**

sameZone(sensor_1,bulb_2) returns **TRUE (room2=room2).**

sameZone(sensor_1,bulb_3) returns **TRUE (room2=room2).**

sameZone(sensor_1,doubleHingedDoor) returns **TRUE (room2=room2).**

sameZone(sensor_1,slidingDoor) returns **FALSE (room2!=room1).**

sameZone(sensor_1,windowBlinds) returns **FALSE (room2!=room1).**

So now the calls to **L5** look like the following:

distance(sensor_1,bulb_1)= ∞

distance(sensor_1,bulb_2)=
Sqrt[(x(sensor_1)-x(bulb_2))^2 + (y(sensor_1)-y(bulb_2))^2]

distance(sensor_1,bulb_3)=
Sqrt[(x(sensor_1)-x(bulb_3))^2 + (y(sensor_1)-y(bulb_3))^2]

distance(sensor_1,doubleHingedDoor)=
Sqrt[(x(sensor_1)-x(doubleHingedDoor))^2 +
(y(sensor_1)-y(doubleHingedDoor))^2]

distance(sensor_1,slidingDoor)= ∞

distance(sensor_1,windowBlinds)= ∞

with

x(sensor_1)=3 y(sensor_1)=4
x(bulb_2)=3 y(bulb_2)=2
x(bulb_3)=4 y(bulb_3)=4
x(doubleHingedDoor)=2 y(doubleHingedDoor)=2
x(slidingDoor)=1 y(slidingDoor)=0
x(windowBlinds)=0 y(windowBlinds)=3

which give

distance(sensor_1,bulb_1)= ∞
distance(sensor_1,bulb_2)=2
distance(sensor_1,bulb_3)=1
distance(sensor_1,doubleHingedDoor)=Sqrt(5)
distance(sensor_1,slidingDoor)= ∞
distance(sensor_1,windowBlinds)= ∞

using these values to evaluate previous **L3** calls gives

directLightExposure(sensor_1,bulb_1)=1750/∞
directLightExposure(sensor_1,bulb_2)=1500/4
directLightExposure(sensor_1,bulb_3)=800/1
directLightExposure(sensor_1,doubleHingedDoor)=1750/5
directLightExposure(sensor_1,slidingDoor)=0/∞
directLightExposure(sensor_1,windowBlinds)=0/∞

Finally the call to L4 for sensor_1 gives

ambientLightIntensity(sensor_1)=0+375+800+350+0+0= 1525

That is equal to the value predicted by the Prediction Model.

This confirm the augmentation in reading value (we saw on **Figure 119**) for sensor_1 when we opened the door, from **1175** to **1525**. Sensor_1 is now exposed to light from the next room.

sensor_2 : According the trace in the figure, the value calculated by **L4** for **sensor_2** should be **757.5**. The 2D Light Law Set calls are as following:

Call to **L4**:

ambientLightIntensity(sensor_2)=
directLightExposure(sensor_2,bulb_1)+
directLightExposure(sensor_2,bulb_2)+
directLightExposure(sensor_2,bulb_3)+
directLightExposure(sensor_2,doubleHingedDoor)+
directLightExposure(sensor_2,slidingDoor)+
directLightExposure(sensor_2,windowBlinds)

- this generates the following calls to the Law **L3**; one for each device

directLightExposure(sensor_2,bulb_1)=
luminousFlux(bulb_1) / distance(sensor_2,bulb_1)^2

directLightExposure(sensor_2,bulb_2)=
luminousFlux(bulb_2) / distance(sensor_2,bulb_2)^2

directLightExposure(sensor_2,bulb_3)=
luminousFlux(bulb_3) / distance(sensor_2,bulb_3)^2

directLightExposure(sensor_2,doubleHingedDoor)=
luminousFlux(doubleHingedDoor) / distance(sensor_2,doubleHingedDoor)^2

directLightExposure(sensor_2,slidingDoor)=
luminousFlux(slidingDoor) / distance(sensor_2,slidingDoor)^2

directLightExposure(sensor_2,windoBlinds)=
luminousFlux(windoBlinds) / distance(sensor_2,windoBlinds)^2

with

luminousFlux(bulb_1)=1750

luminousFlux(bulb_2)=1500 (CFL reached its maximum capacity)

luminousFlux(bulb_3)=800

luminousFlux(doubleHingedDoor)=1*ambientLightIntensity(doubleHingedDoor)=0

because **doubleHingedDoor.AmbientLightModificationRatio=1** (door opened).

Other Modifiers are kept closed:

luminousFlux(slidingDoor)=0

luminusFlux(windoBlinds)=0

Now that the double hinged door is fully opened, we have to evaluate

ambientLightIntensity(doubleHingedDoor)in room2 to estimate how much the doubleHingedDoor contributes in luminous flux in room1

So:

luminousFlux(doubleHingedDoor)=1*ambientLightIntensity(doubleHingedDoor)
in room1 in room2

The call is treated as if it was a call from a sensor, but with ignoring the effect modifier doubleHingedDoor in the calculations.

So, unlike when sensor_1 called **ambientLightIntensity** in room2, when the doubleHingedDoor calls it, it generates 5 calls to directLightExposure instead of 6 calls. These calls are highlighted in blue:

doubleHingedDoor :

Call to **L4**:

```
ambientLightIntensity(doubleHingedDoor)=
directLightExposure(doubleHingedDoor,bulb_1)+
directLightExposure(doubleHingedDoor,bulb_2)+
directLightExposure(doubleHingedDoor,bulb_3)+
directLightExposure(doubleHingedDoor,doubleHingedDoor)+
directLightExposure(doubleHingedDoor,slidingDoor)+
directLightExposure(doubleHingedDoor,windowBlinds)
```

- this generates the following calls to the Law **L3**; one for each device

```
directLightExposure(doubleHingedDoor,bulb_1)=
luminousFlux(bulb_1) / distance(doubleHingedDoor,bulb_1)^2

directLightExposure(doubleHingedDoor,bulb_2)=
luminousFlux(bulb_2) / distance(doubleHingedDoor,bulb_2)^2

directLightExposure(doubleHingedDoor,bulb_3)=
luminousFlux(bulb_3) / distance(doubleHingedDoor,bulb_3)^2

directLightExposure(doubleHingedDoor,doubleHingedDoor)=0

directLightExposure(doubleHingedDoor,slidingDoor)=
luminousFlux(slidingDoor) / distance(doubleHingedDoor,slidingDoor)^2

directLightExposure(doubleHingedDoor,windoBlinds)=
luminousFlux(windoBlinds) / distance(doubleHingedDoor,windoBlinds)^2
```

with

```
luminousFlux(bulb_1)=1750
luminousFlux(bulb_2)=1500
luminousFlux(bulb_3)=800
luminousFlux(slidingDoor)=0
luminusFlux(windoBlinds)=0
```

(Closed Effect Modifiers):

- this generates the following calls to the Law **L5**; one for each component
- before evaluating **L5**, each **L5** generates the following calls to **L1** (the same-zone verification function)

doubleHingedDoor has room1 and room2 as Zones in property, but in these calculations it is considered as part of room1 (we want to calculate how much of light from room1 is going to room2).

sameZone returns FALSE when comparing zones of same objects for additional safety (so the Effect Modifier doesn't consider itself in the calculations).

sameZone(doubleHingedDoor,bulb_1) returns **FALSE (room2!=room1).**

sameZone(doubleHingedDoor,bulb_2) returns **TRUE (room2=room2).**

sameZone(doubleHingedDoor,bulb_3) returns **TRUE (room2=room2).**

sameZone(doubleHingedDoor,doubleHingedDoor) returns **FALSE.**

sameZone(doubleHingedDoor,slidingDoor) returns **FALSE (room2!=room1).**

sameZone(doubleHingedDoor,windowBlinds) returns **FALSE (room2!=room1).**

So now the calls to **L5** look like the following:

distance(doubleHingedDoor,bulb_1)= ∞

distance(doubleHingedDoor,bulb_2)=
Sqrt[(x(doubleHingedDoor)-x(bulb_2))^2+
(y(doubleHingedDoor)-y(bulb_2))^2]

distance(doubleHingedDoor,bulb_3)=
Sqrt[(x(doubleHingedDoor)-x(bulb_3))^2+
(y(doubleHingedDoor)-y(bulb_3))^2]

distance(doubleHingedDoor,doubleHingedDoor)= ∞ (also for additional safety)

distance(doubleHingedDoor,slidingDoor)= ∞

distance(doubleHingedDoor,windowBlinds)= ∞

with

x(doubleHingedDoor)=2 y(doubleHingedDoor)=2
x(bulb_2)=3 y(bulb_2)=2
x(bulb_3)=4 y(bulb_3)=4
x(slidingDoor)=1 y(slidingDoor)=0
x(windowBlinds)=0 y(windowBlinds)=3

which give

distance(doubleHingedDoor,bulb_1)= ∞
distance(doubleHingedDoor,bulb_2)=1
distance(doubleHingedDoor,bulb_3)=Sqrt(8)
distance(doubleHingedDoor,doubleHingedDoor)= ∞
distance(doubleHingedDoor,slidingDoor)= ∞
distance(doubleHingedDoor,windowBlinds)= ∞

using these values to evaluate previous **L3** calls gives

directLightExposure(doubleHingedDoor,bulb_1)=1750/∞
directLightExposure(doubleHingedDoor,bulb_2)=1500/1
directLightExposure(doubleHingedDoor,bulb_3)=800/8
directLightExposure(doubleHingedDoor,doubleHingedDoor)=0
directLightExposure(doubleHingedDoor,slidingDoor)=0
directLightExposure(doubleHingedDoor,windowBlinds)=0

Finally the call to L4 for sensor_1 gives

ambientLightIntensity(doubleHingedDoor)=0+1500+100+0+0+0= 1600

So **luminousFlux(doubleHingedDoor)=1*1600=1600**

Now that we know the value of **luminousFlux(doubleHingedDoor)=1600**, we continue our calculations as if the doubleHingedDoor was another actuator in room2.

We return to Room2.

We left at estimating **directLightExposure** for every component.

- this generates the following calls to the Law **L5**; one for each component
- before evaluating **L5**, each **L5** generates the following calls to **L1** (the same-zone verification function)

sameZone(sensor_2,bulb_1) returns **TRUE (room1=room1).**

sameZone(sensor_2,bulb_2) returns **FALSE (room1!=room2).**

sameZone(sensor_2,bulb_3) returns **FALSE (room1!=room2).**

sameZone(sensor_2,doubleHingedDoor) returns **TRUE (room1=room1).**

sameZone(sensor_2,slidingDoor) returns **TRUE (room1=room1).**

sameZone(sensor_2,windowBlinds) returns **TRUE (room1=room1).**

So now the calls to **L5** look like the following:

distance(sensor_2,bulb_1)=
Sqrt[(x(sensor_2)-x(bulb_1))^2 + (y(sensor_2)-y(bulb_1))^2]

distance(sensor_2,bulb_2)= ∞

distance(sensor_2,bulb_3)= ∞

distance(sensor_2,doubleHingedDoor)=
Sqrt[(x(sensor_2)-x(doubleHingedDoor))^2 +
(y(sensor_2)-y(doubleHingedDoor))^2]

```
distance(sensor_2,slidingDoor)=
Sqrt[(x(sensor_2)-x(slidingDoor))^2 +
(y(sensor_2)-y(slidingDoor))^2]

distance(sensor_2,windowBlinds)=
Sqrt[(x(sensor_2)-x(windowBlinds))^2 +
(y(sensor_2)-y(windowBlinds))^2]
```

with

```
x(sensor_2)=1 y(sensor_2)=4
x(bulb_1)=1 y(bulb_1)=2
x(doubleHingedDoor)=2 y(doubleHingedDoor)=2
x(slidingDoor)=1 y(slidingDoor)=0
x(windowBlinds)=0 y(windowBlinds)=3
```

which give

```
distance(sensor_2,bulb_1)=2
distance(sensor_2,bulb_2)= ∞
distance(sensor_2,bulb_3)= ∞
distance(sensor_2,doubleHingedDoor)=Sqrt(5)
distance(sensor_2,slidingDoor)=4
distance(sensor_2,windowBlinds)=Sqrt(2)
```

using these values to evaluate previous **L3** calls gives

```
directLightExposure(sensor_2,bulb_1)=1750/4
directLightExposure(sensor_2,bulb_2)=1500/∞
directLightExposure(sensor_2,bulb_3)=800/∞
directLightExposure(sensor_2,doubleHingedDoor)=1600/5
directLightExposure(sensor_2,slidingDoor)=0
directLightExposure(sensor_2,windowBlinds)=0
```

Finally the call to L4 for sensor_1 gives

```
ambientLightIntensity(sensor_1)=437.5+0+0+320+0+0= 757.5
```

That is equal to the value predicted by the Prediction Model.

This confirm the augmentation in reading value (we saw on **Figure 119**) for sensor_1 when we opened the door, from **437.5** to **1195**. Sensor_1 is now exposed to light from the next room.

Test 3:

Evaluating the Mathematical Model and Validation for results reported in **Figure 121**:

sensor_2 : According the trace in the figure, the value calculated by **L4** for **sensor_2** should be **16937.5**. The 2D Light Law Set calls are as following:

Call to **L4**:

```
ambientLightIntensity(sensor_2)=
directLightExposure(sensor_2,bulb_1)+
directLightExposure(sensor_2,bulb_2)+
directLightExposure(sensor_2,bulb_3)+
directLightExposure(sensor_2,doubleHingedDoor)+
directLightExposure(sensor_2,slidingDoor)+
directLightExposure(sensor_2,windowBlinds)
```

- this generates the following calls to the Law **L3**; one for each device

```
directLightExposure(sensor_2,bulb_1)=
luminousFlux(bulb_1) / distance(sensor_2,bulb_1)^2

directLightExposure(sensor_2,bulb_2)=
luminousFlux(bulb_2) / distance(sensor_2,bulb_2)^2

directLightExposure(sensor_2,bulb_3)=
luminousFlux(bulb_3) / distance(sensor_2,bulb_3)^2

directLightExposure(sensor_2,doubleHingedDoor)=
luminousFlux(doubleHingedDoor) / distance(sensor_2,doubleHingedDoor)^2

directLightExposure(sensor_2,slidingDoor)=
luminousFlux(slidingDoor) / distance(sensor_2,slidingDoor)^2

directLightExposure(sensor_2,windoBlinds)=
luminousFlux(windoBlinds) / distance(sensor_2,windoBlinds)^2
```

with

luminousFlux(bulb_1)=1750

luminousFlux(bulb_2)=1500 (CFL reached its maximum capacity)

luminousFlux(bulb_3)=800

luminousFlux(doubleHingedDoor)=0

luminousFlux(slidingDoor)=0

luminusFlux(windoBlinds)=1*AmbientLight(trustedSensor)

The value of the AmbientLight property of the trusted sensor for windowBlinds (this is the special case discussed in Effect Modifier secion in **4.5**), which is **sensor_3** is **33000**.

So:

```
luminusFlux(windoBlinds)=33000
```

- this generates the following calls to the Law **L5**; one for each component
- before evaluating **L5**, each **L5** generates the following calls to **L1** (the same-zone verification function)

sameZone(sensor_2,bulb_1) returns **TRUE (room1=room1).**

sameZone(sensor_2,bulb_2) returns **FALSE (room1!=room2).**

sameZone(sensor_2,bulb_3) returns **FALSE (room1!=room2).**

sameZone(sensor_2,doubleHingedDoor) returns **TRUE (room1=room1).**

sameZone(sensor_2,slidingDoor) returns **TRUE (room1=room1).**

sameZone(sensor_2,windowBlinds) returns **TRUE (room1=room1).**

So now the calls to **L5** look like the following:

distance(sensor_2,bulb_1)=
Sqrt[(x(sensor_2)-x(bulb_1))^2 + (y(sensor_2)-y(bulb_1))^2]

distance(sensor_2,bulb_2)= ∞

distance(sensor_2,bulb_3)= ∞

distance(sensor_2,doubleHingedDoor)=
Sqrt[(x(sensor_2)-x(doubleHingedDoor))^2 +
(y(sensor_2)-y(doubleHingedDoor))^2]

distance(sensor_2,slidingDoor)=
Sqrt[(x(sensor_2)-x(slidingDoor))^2 +
(y(sensor_2)-y(slidingDoor))^2]

distance(sensor_2,windowBlinds)=
Sqrt[(x(sensor_2)-x(windowBlinds))^2 +
(y(sensor_2)-y(windowBlinds))^2]

with

x(sensor_2)=1 y(sensor_2)=4
x(bulb_1)=1 y(bulb_1)=2
x(doubleHingedDoor)=2 y(doubleHingedDoor)=2
x(slidingDoor)=1 y(slidingDoor)=0
x(windowBlinds)=0 y(windowBlinds)=3

which give

distance(sensor_2,bulb_1)=2
distance(sensor_2,bulb_2)= ∞
distance(sensor_2,bulb_3)= ∞
distance(sensor_2,doubleHingedDoor)=Sqrt(5)

distance(sensor_2,slidingDoor)=4
distance(sensor_2,windowBlinds)=Sqrt(2)

using these values to evaluate previous **L3** calls gives

directLightExposure(sensor_2,bulb_1)=1750/4
directLightExposure(sensor_2,bulb_2)=1500/∞
directLightExposure(sensor_2,bulb_3)=800/∞
directLightExposure(sensor_2,doubleHingedDoor)=0
directLightExposure(sensor_2,slidingDoor)=0
directLightExposure(sensor_2,windowBlinds)=33000/2

Finally the call to L4 for sensor_1 gives

ambientLightIntensity(sensor_1)=437.5+0+0+0+0+16500= 16937.5

That is equal to the value predicted by the Prediction Model.

Test 4:

Evaluating the Mathematical Model and Validation for results reported in **Figure 122**:

sensor_1 : According the trace in the figure, the value calculated by **L4** for **sensor_1** should be **2450**. The 2D Light Law Set calls are as following:

Call to **L4**:

```
ambientLightIntensity(sensor_1)=
directLightExposure(sensor_1,bulb_1)+
directLightExposure(sensor_1,bulb_2)+
directLightExposure(sensor_1,bulb_3)+
directLightExposure(sensor_1,doubleHingedDoor)+
directLightExposure(sensor_1,slidingDoor)+
directLightExposure(sensor_1,windowBlinds)
```

- this generates the following calls to the Law **L3**; one for each device

```
directLightExposure(sensor_1,bulb_1)=
luminousFlux(bulb_1) / distance(sensor_1,bulb_1)^2

directLightExposure(sensor_1,bulb_2)=
luminousFlux(bulb_2) / distance(sensor_1,bulb_2)^2

directLightExposure(sensor_1,bulb_3)=
luminousFlux(bulb_3) / distance(sensor_1,bulb_3)^2

directLightExposure(sensor_1,doubleHingedDoor)=
luminousFlux(doubleHingedDoor) / distance(sensor_1,doubleHingedDoor)^2

directLightExposure(sensor_1,slidingDoor)=
```

luminousFlux(slidingDoor) / distance(sensor_1,slidingDoor)^2

directLightExposure(sensor_1,windoBlinds)=
luminousFlux(windoBlinds) / distance(sensor_1,windoBlinds)^2

with

luminousFlux(bulb_1)=1750

luminousFlux(bulb_2)=1500 (CFL reached its maximum capacity)

luminousFlux(bulb_3)=800

luminousFlux(doubleHingedDoor)=1*ambientLightIntensity(doubleHingedDoor)=0.5

because **doubleHingedDoor.AmbientLightModificationRatio=0.5** (half opened).

Other Modifiers are kept closed:

luminousFlux(slidingDoor)=0

luminusFlux(windoBlinds)=1*AmbientLight(trustedSensor)

The value of the AmbientLight property of the trusted sensor for windowBlinds (this is the special case discussed in Effect Modifier secion in **4.5**), which is **sensor_3** is **33000**.

So:

luminusFlux(windoBlinds)=33000

Now that the double hinged door is half opened, we have to evaluate

ambientLightIntensity(doubleHingedDoor)in room1 to estimate how much the doubleHingedDoor contributes in luminous flux in room2

So:

luminousFlux(doubleHingedDoor)=0.5*ambientLightIntensity(doubleHingedDoor)
in room2 in room1

The call is treated as if it was a call from a sensor, but with ignoring the effect modifier doubleHingedDoor in the calculations.

So, unlike when sensor_2 called **ambientLightIntensity** in room1, when the doubleHingedDoor calls it, it generates 5 calls to directLightExposure instead of 6 calls. These calls are highlighted in blue:

doubleHingedDoor :

Call to **L4**:

ambientLightIntensity(doubleHingedDoor)=
directLightExposure(doubleHingedDoor,bulb_1)+
directLightExposure(doubleHingedDoor,bulb_2)+
directLightExposure(doubleHingedDoor,bulb_3)+
directLightExposure(doubleHingedDoor,doubleHingedDoor)+
directLightExposure(doubleHingedDoor,slidingDoor)+
directLightExposure(doubleHingedDoor,windowBlinds)

- this generates the following calls to the Law **L3**; one for each device

directLightExposure(doubleHingedDoor,bulb_1)=
luminousFlux(bulb_1) / distance(doubleHingedDoor,bulb_1)^2

directLightExposure(doubleHingedDoor,bulb_2)=
luminousFlux(bulb_2) / distance(doubleHingedDoor,bulb_2)^2

directLightExposure(doubleHingedDoor,bulb_3)=
luminousFlux(bulb_3) / distance(doubleHingedDoor,bulb_3)^2

directLightExposure(doubleHingedDoor,doubleHingedDoor)=0

directLightExposure(doubleHingedDoor,slidingDoor)=
luminousFlux(slidingDoor) / distance(doubleHingedDoor,slidingDoor)^2

directLightExposure(doubleHingedDoor,windoBlinds)=
luminousFlux(windoBlinds) / distance(doubleHingedDoor,windoBlinds)^2

with

luminousFlux(bulb_1)=1750
luminousFlux(bulb_2)=1500
luminousFlux(bulb_3)=800
luminousFlux(slidingDoor)=0
luminusFlux(windoBlinds)=0

(Closed Effect Modifiers):

- this generates the following calls to the Law **L5**; one for each component
- before evaluating **L5**, each **L5** generates the following calls to **L1** (the same-zone verification function)

doubleHingedDoor has room1 and room2 as Zones in property, but in these calculations it is considered as part of room1 (we want to calculate how much of light from room1 is going to room2).

sameZone returns FALSE when comparing zones of same objects for additional safety (so the Effect Modifier doesn't consider itself in the calculations).

sameZone(doubleHingedDoor,bulb_1) returns **TRUE (room1=room1).**

sameZone(doubleHingedDoor,bulb_2) returns **FALSE (room1!=room2).**

sameZone(doubleHingedDoor,bulb_3) returns **FALSE (room1!=room2).**

sameZone(doubleHingedDoor,doubleHingedDoor) returns **FALSE.**

sameZone(doubleHingedDoor,slidingDoor) returns **TRUE (room1=room1).**

sameZone(doubleHingedDoor,windowBlinds) returns **TRUE (room1=room1).**

So now the calls to **L5** look like the following:

distance(doubleHingedDoor,bulb_1)=
Sqrt[(x(doubleHingedDoor)-x(bulb_1))^2+
(y(doubleHingedDoor)-y(bulb_1))^2]

distance(doubleHingedDoor,bulb_2)= ∞

distance(doubleHingedDoor,bulb_3)= ∞

distance(doubleHingedDoor,doubleHingedDoor)= ∞ (also for additional safety)

distance(doubleHingedDoor,slidingDoor)=
Sqrt[(x(doubleHingedDoor)-x(slidingDoor))^2 +
(y(doubleHingedDoor)-y(slidingDoor))^2]

distance(doubleHingedDoor,windowBlinds)=
Sqrt[(x(doubleHingedDoor)-x(windowBlinds))^2 +
(y(doubleHingedDoor)-y(windowBlinds))^2]

with

x(doubleHingedDoor)=2 y(doubleHingedDoor)=2
x(bulb_1)=1 y(bulb_1)=2
x(slidingDoor)=1 y(slidingDoor)=0
x(windowBlinds)=0 y(windowBlinds)=3

which give

distance(doubleHingedDoor,bulb_1)=1
distance(doubleHingedDoor,bulb_2)= ∞
distance(doubleHingedDoor,bulb_3)= ∞
distance(doubleHingedDoor,doubleHingedDoor)= ∞
distance(doubleHingedDoor,slidingDoor)=Sqrt(3)
distance(doubleHingedDoor,windowBlinds)=Sqrt(3)

using these values to evaluate previous **L3** calls gives

directLightExposure(doubleHingedDoor,bulb_1)=1750/1
directLightExposure(doubleHingedDoor,bulb_2)=1500/∞
directLightExposure(doubleHingedDoor,bulb_3)=800/∞
directLightExposure(doubleHingedDoor,doubleHingedDoor)=0
directLightExposure(doubleHingedDoor,slidingDoor)=0/3
directLightExposure(doubleHingedDoor,windowBlinds)=33000/3

Finally the call to L4 for sensor_1 gives

ambientLightIntensity(doubleHingedDoor)=1750+0+0+0+0+11000= 12750

So **luminousFlux(doubleHingedDoor)=0.5*12750=6375**

Now that we know the value of **luminousFlux(doubleHingedDoor)=6375**, we continue our calculations as if the doubleHingedDoor was another actuator in room2.

We return to Room2.

We left at estimating **directLightExposure** for every component.

- this generates the following calls to the Law **L5**; one for each component
- before evaluating **L5**, each **L5** generates the following calls to **L1** (the same-zone verification function)

sameZone(sensor_1,bulb_1) returns **FALSE (room2!=room1).**

sameZone(sensor_1,bulb_2) returns **TRUE (room2=room2).**

sameZone(sensor_1,bulb_3) returns **TRUE (room2=room2).**

sameZone(sensor_1,doubleHingedDoor) returns **TRUE (room2=room2).**

sameZone(sensor_1,slidingDoor) returns **FALSE (room2!=room1).**

sameZone(sensor_1,windowBlinds) returns **FALSE (room2!=room1).**

So now the calls to **L5** look like the following:

distance(sensor_1,bulb_1)= ∞

distance(sensor_1,bulb_2)=
Sqrt[(x(sensor_1)-x(bulb_2))^2 + (y(sensor_1)-y(bulb_2))^2]

distance(sensor_1,bulb_3)=
Sqrt[(x(sensor_1)-x(bulb_3))^2 + (y(sensor_1)-y(bulb_3))^2]

distance(sensor_1,doubleHingedDoor)=
Sqrt[(x(sensor_1)-x(doubleHingedDoor))^2 +
(y(sensor_1)-y(doubleHingedDoor))^2]

distance(sensor_1,slidingDoor)= ∞

distance(sensor_1,windowBlinds)= ∞

with

x(sensor_1)=3 y(sensor_1)=4
x(bulb_2)=3 y(bulb_2)=2
x(bulb_3)=4 y(bulb_3)=4
x(doubleHingedDoor)=2 y(doubleHingedDoor)=2
x(slidingDoor)=1 y(slidingDoor)=0
x(windowBlinds)=0 y(windowBlinds)=3

which give

distance(sensor_1,bulb_1)= ∞
distance(sensor_1,bulb_2)=2
distance(sensor_1,bulb_3)=1
distance(sensor_1,doubleHingedDoor)=Sqrt(5)
distance(sensor_1,slidingDoor)= ∞
distance(sensor_1,windowBlinds)= ∞

using these values to evaluate previous **L3** calls gives

directLightExposure(sensor_1,bulb_1)=1750/∞
directLightExposure(sensor_1,bulb_2)=1500/4
directLightExposure(sensor_1,bulb_3)=800/1
directLightExposure(sensor_1,doubleHingedDoor)=6375/5
directLightExposure(sensor_1,slidingDoor)=0/∞
directLightExposure(sensor_1,windowBlinds)=0/∞

Finally the call to L4 for sensor_1 gives

ambientLightIntensity(sensor_1)=0+375+800+1275+0+0= 2450

That is equal to the value predicted by the Prediction Model.

Annex-D

Rule1:

```
[ CMD_LIGHT_ON_1:

    (?MS RDF:type AMI:PresenceSensor),
    (?LA RDF:type AMI:LightActuator),
    (?R RDF:type ?RT),
    (?RT RDFS:subClassOf AMI:Room),
    (?MS AMI:isIn ?R),
    (?LA AMI:isIn ?R),
    (?MS CORE:realStateStringValue 'personInside'),
    (?F RDF:type AMI:OnOffFunctionality),
    (?LA AMI:hasFunctionality ?F),
    (?C RDF:type AMI:OnCommand)

          -> (?F AMI:hasCommand ?C) ]
```

Rule2:

```
[ CMD_LIGHT_ON_2:

    (?MS RDF:type AMI:PresenceSensor),
    (?LS RDF:type AMI:LightSensor),
    (?LA RDF:type AMI:LightActuator),
    (?R RDF:type ?RT),
    (?RT RDFS:subClassOf AMI:Room),
    (?MS AMI:isIn ?R),
    (?LS AMI:isIn ?R),
    (?LA AMI:isIn ?R),
    (?DP RDF:type AMI:AAL_DP),
    (?DP AMI:isCurentDP ?curDP),
    equal(?curDP,'true'),
    (?DP AMI:low_AAL_LightThreshold ?LLT),
    (?MS CORE:realStateStringValue 'personInside'),
    (?LS AMI:realIntValue ?LMV),
    lessThan(?LMV,?LLT),
    (?F RDF:type AMI:OnOffFunctionality),
    (?LA AMI:hasFunctionality ?F),
    (?C RDF:type AMI:OnCommand)

          -> (?F AMI:hasCommand ?C) ]
```

References

[1] D. Estrin, D. Culler, K. Pister, and G. Sukhatme, "Connecting the Physical World with Pervasive Networks", IEEE Pervasive Computing, pp.59-69. 2002.

[2] Dorf, Richard C., and Robert H. Bishop. Modern control systems. Pearson, 2011.

[3] Menno Lindwer, Diana Marculescu, Twan Basten, Rainer Zimmermann, Radu Marculescu, Stefan Jung, Eugenio Cantatore – "Ambient Intelligence Visions and Achievements: Linking Abstract Ideas to Real-World Concepts." Automation and Test in Europe Conference and Exhibition. IEEE, pp.10-15. 2003.

[4] Ramos, C., Augusto, J. C., and Shapiro, D. Ambient intelligence - the next step for artificial intelligence. IEEE Intelligent Systems, vol.23, no.2 , pp.15-18. 2008.

[5] C. Ramos, "Ambient Intelligence - a State of the Art from Artificial Intelligence Perspective," Proc. 13th Portuguese Conf. Ar-tificial Intelligence Workshops, LNAI 4874, Springer, pp.285-295. 2007.

[6] J.C. Augusto and C.D. Nugent, eds., Designing Smart Homes: The Role of Artificial Intelligence, LNAI, Volume 4008, Springer, 2006.

[7] Ascencio, R.R.L.; Aguilera Galicia, C., "Biomass estimation using artificial neural networks on field programmable analog devices", 2000. ISIE 2000. Proceedings of the 2000 IEEE International Symposium on Industrial Electronics, vol.1, no., pp.61-66 vol.1, 2000.

[8] Hollis, Paul W., and John J. Paulos. "Artificial neural networks using MOS analog multipliers." Solid-State Circuits, IEEE Journal of 25, no. 3, pp.849-855. 1990.

[9] Papert, S., Watt, D ., diSessa, A ., & Weir, S. "Final report of the Brookline LOGO Project: An assessment and documentation of a children's computer laboratory." Cambridge, MA: MIT Division for Study and Research in Education, 1979.

[10] Yu, Victor L., Lawrence M. Fagan, Sharon M. Wraith, William J. Clancey, A. Carlisle Scott, John Hannigan, Robert L. Blum, Bruce G. Buchanan, and Stanley N. Cohen. "Antimicrobial selection by a computer: a blinded evaluation by infectious disease experts". Journal of the American Medical Association vol.242, no.12, pp.1279-1282. doi:10.1001/jama.1979.03300120033020. PMID 480542. 1979.

[11] R. Fikes and N. Nilsson. "STRIPS: a new approach to the application of theorem proving to problem solving." Artificial Intelligence, 2, pp.189-208.1971.

[12] Dzierzanowski, J. M., Chrisman, K. R., MacKinnon, G. J., & Klahr, "The Authorizer's Assistant. A Knowledge-based Credit Authorization System for American Express". Proceedings of the Conference on innovative Applications of Artificial Intelligence, AAAI, 1989.

[13] Francesco Ricci and Lior Rokach and Bracha Shapira, Introduction to Recommender Systems Handbook, Recommender Systems Handbook, Springer, 2011, pp. 1-35.

[14] M. Weiser. Hot topics: Ubiquitous computing. IEEE Computer, vol.26, no.10, pp.71-72, 1993.

[15] Michael Coen, "Design Principals for Intelligent Environments", Intelligent Environments, Papers from the 1998 AAAI Spring Symposium, Technical Report SS-98-02, AAAI Press. 23-25 March, 1998.

[16] Cohen, P., "The role of natural language in a multimodal interface" Proceedings of User Interface Software Technology (UIST'92) Conference, Academic Press, Monterey, California, pp.143-149. 1992.

[17] Oviatt, S.L., De Angeli, A., Kuhn, K., "Integration and Synchronization of Input Modes during Multimodal Human-Computer Interaction". In Proceedings of CHI 1997, pp.415-422, 1997.
[18] E. Aarts and J. Encarnacao. "True Visions: The Emergence of Ambient Intelligence". Springer, 2006.
[19] Remagnino, Paolo, and Gian Luca Foresti. « Ambient intelligence: A new multidisciplinary paradigm. "Systems, Man and Cybernetics, Part A: Systems and Humans", IEEE Transactions on 35.1, pp.1-6. 2005
[20] Augusto, J.C., Nakashima, H., Aghajan, H. Handbook on Ambient Intelligence and Smart Environments: a state of the art. Springer Verlag 2009.
[21] Wooldridge, Michael. An Introduction to MultiAgent Systems. John Wiley & Sons. ISBN 0-471-49691-X. pp. 366. 2002.
[22] M. Weiser. The computer for the twenty-first century. Scientific American, 165: pp.94-104, 1991.
[23] DJ Cook, JC Augusto, and VR Jakkulaa, "Ambient intelligence: Technologies, applications, and opportunities," Pervasive and Mobile Computing, vol. Volume 5, Issue 4, pp. 277-298, 2009.
[24] K. Ducatel, M. Bogdanowicz, F. Scapolo, J. Leijten, and J.- C. Burgelman, "Scenarios for ambient intelligence in 2010," European Commission, Tech. Rep., 2001.
[25] C. K. M. Crutzen. Invisibility and the meaning of ambient intelligence. International Review of Information Ethics, 6: pp.1-11, 2006.
[26] J. Rech and K.-D.Althoff. Artificial intelligence and software engineering: Status and future trends. Themenschwerpunkt KI & SE, KI, 3:5-11, 2004.
[27] A. Vasilakos and W. Pedrycz. "Ambient Intelligence, Wireless Networking, and Ubiquitous Computing". Artech House Publishers, 2006.
[28] Raffler, H. "Other perspectives on ambient intelligence." Password Mag 2006.
[29] Kephart, Jeffrey O., and David M. Chess. "The vision of autonomic computing." Computer 36.1, pp.41-50. 2003.
[30] A. Dey. Understanding and Using Context. Personal and Ubiquitous Computing, vol.5, no.1, pp.4-7, 2001.
[31] Zhang, W., Hansen, K.M.: Semantic web based self-management for a pervasive service middleware. In: Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2008), Venice, Italy, pp.245-254. October 2008.
[32] P. Krill. IBM research envisions pervasive computing. Info World, 2000.
[33] E. Maeda and Y. Minami. "Steps toward ambient intelligence". NIT Technical Review vol.4, no.1, 2006.
[34] Brammer, Karl, and Gerhard Siffling. "Kalman-bucy filters". Norwood: Artech House, 1989.
[35] J. O. Berger. "Statistical Decisions." Springer-Verlag, 1985.
[36] J. Manyika and H. Durrant-Whyte. "Data Fusion and Sensor Management: A Decentralized Information-Theoretic Approach". Ellis Horwood, 1994.
[37] R. Xu, G. Mei, Z. Ren, C. Kwan, J. Aube, C. Rochet, and V. Stanford. Speaker identification and speech recognition using phased arrays. In Y. Cai and J. Abascal, editors, Ambient Intelligence in Everyday Life, pp.227-238. Springer, 2006.
[38] Cook, Diane, and Sajal Das. "Smart environments: Technology, protocols and applications." Vol. 43. John Wiley & Sons, 2004.
[39] M. Pantic. Face for ambient intelligence. In Y. Cai and J. Abascal, editors, "Ambient Intelligence in Everyday Life", pp.32-66. Springer, 2006.

[40] R. Nakatsu. “Integration of multimedia and art for new human-computer communications”. In Proceedings of the Pacific Rim International Conference on Artificial Intelligence, pp.19-28, 2002.

[41] Tao, Jianhua; Tieniu Tan. “Affective Computing: A Review". Affective Computing and Intelligent Interaction”. LNCS 3784. Springer, pp.981-995. 2005.

[42] F. Wang et al., “Social Computing: From Social Informatics to Social Intelligence”. IEEE Intelligent Systems, vol. 22, no. 2, pp.79-83, 2007.

[43] Shan, Caifeng, and Ralph Braspenning. “Recognizing facial expressions automatically from video.” Handbook of ambient intelligence and smart environments. Springer US, pp.479-509. 2010

[44] Aghajan, Yasmin, et al. “Home Exercise in a Social Context: Real-Time Experience Sharing Using Avatars.” Intelligent Technologies for Interactive Entertainment. Springer Berlin Heidelberg, pp.19-31. 2009.

[45] Aghajan, Hamid, Juan Carlos Augusto, and Ramón López-Cózar Delgado, eds. “Human-centric interfaces for ambient intelligence”. Academic Press, 2009.

[46] Jaimes, Alejandro, Nicu Sebe, and Daniel Gatica-Perez. "Human-centered computing: a multimedia perspective”. Proceedings of the 14th annual ACM international conference on Multimedia. ACM, pp.255-864. 2006.

[47] David Canfield Smith, “Pygmalion: a creative programming environment”, Report no. STAN-CS-75-499. Stanford U. 1975.

[48] R. S. Amant, H. Lieberman, R. Potter, and L. Zettlemoyer, “Programming by example: visual generalization in programming by example,” Commun. ACM, vol. 43, no. 3, pp.107-114, 2000.

[49] Chin, J., Callaghan, V., Clarke, G., “Soft-appliances: A vision for user created networked appliances in digital homes”, Journal of Ambient Intelligence and Smart Environments 1, pp.69-75. 2009.

[50] Chin, J. S. Y., V. Callaghan, M. Colley, H. Hagras, and G. Clarke. “Virtual appliances for pervasive computing: A deconstructionist, ontology based, programming-by-example approach.” v2-152. 2005.

[51] Jeannette S Chin, Vic Callaghan, and Graham Clarke. “An end-user programming paradigm for pervasive computing applications”. In ICPS’06 : Proceedings of the 3rd International Conference on Pervasive Services, Washington, DC, USA, 2006. doi : 10.1109/PERSER.2006.1652254, pp.325-328. 2006.

[52] Quincy, “The invention of the first wearable computer”, in The Second International Symposium on Wearable Computers: Digest of Papers, IEEE Computer Society, pp.4-8. 1998.

[53] Rousseau, Cyril, Yacine Bellik, Frédéric Vernier, and Didier Bazalgette. “A framework for the intelligent multimodal presentation of information”. Signal Processing vol. 86, no.12, pp.3696-3713. 2006.

[54] Jacquet, Christophe, Yolaine Bourda, and Yacine Bellik. “A context-aware locomotion assistance device for the blind”. In People and Computers XVIII—Design for Life, pp.315-328. Springer London, 2005.

[55] Rousseau, Cyril, Yacine Bellik, and Frédéric Vernier. "Multimodal output specification/simulation platform." In Proceedings of the 7th international conference on Multimodal interfaces, pp.84-91. ACM, 2005.

[56] Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonckt, J. “A unifying reference framework for multi-target user interfaces”. Interacting With Computer, vol.15, no.3, pp.289-308. 2003.

[57] Mark Weiser. “Some computer science issues in ubiquitous computing”. In special issue, Computer Augmented Environments. CACM, vol.36, no.7, pp.74-83, July 1993.

[58] Streitz, Norbert A. "From human–computer interaction to human–environment interaction: Ambient intelligence and the disappearing computer." In Universal Access in Ambient Intelligence Environments, pp.3-13. Springer Berlin Heidelberg, 2007.

[59] Mitleton-Kelly, Eve, et al. "Enhancing Crowd Evacuation and Traffic Management Through AmI Technologies: A Review of the Literature." Co-evolution of Intelligent Socio-technical Systems. Springer Berlin Heidelberg, pp.19-41. 2013.

[60] J. C. Augusto and P. McCullah. Ambient intelligence: Concepts and applications. International Journal on Computer Science and Information Systems, vol.4, no.1: pp.1-28, 2007.

[61] Tolmie, Peter, James Pycock, Tim Diggins, Allan MacLean, and Alain Karsenty. “Unremarkable computing.” In Proceedings of the SIGCHI conference on Human factors in computing systems, pp.399-406. ACM, 2002

[62] D.-M. Han and J.-H. Lim, “Design and implementation of smart home energy management systems based on zigbee” Consumer Electronics, IEEE Transactions on, vol. 56, no. 3, pp.1417-1425. 2010.

[63] DJ Cook, “MavHome: An Agent-Based Smart Home,” in Proc. of 1st IEEE Int. Conf. Pervasive Computing and Communications (PerCom'03). pp.521-524. 2003.

[64] S. K Das, N. Roy and A. Roy “Context-Aware Resource Management in Multi-Inhabitant Smart Homes: A Framework based on Nash H-Learning”, Pervasive and Mobile Computing (PMC) Journal, Vol. 2, Issue 4, pp.372-404. Nov. 2006.

[65] Blatt, Mark N. “Mobile Healthcare Solutions and Innovation in Healthcare”. In proceedings. Brunswick East, Vic.: Health Informatics Society of Australia (HISA) ; Royal Australian College of General Practitioners (RACGP): pp320-321. 2003.

[66] Blatt, Mark, and George Margelis. “Mobile point of care (MPOC) success stories from around the world.” HIC 2010: Proceedings; 18th Annual Health Informatics Conference: Informing the Business of Healthcare, Melbourne Convention and Exhibition Centre. Health Informatics Society of Australia, pp.24-26 August 2010.

[67] Eleni Stroulia , David Chodos , Nicholas M. Boers , Jianzhao Huang , Pawel Gburzynski , Ioanis Nikolaidis, “Software engineering for health education and care delivery systems: The Smart Condo project”. Proceedings of the 2009 ICSE Workshop on Software Engineering in Health Care, pp.20-28, 18-19 May 2009.

[68] Das, S. K. and Cook, D. J. “Health monitoring in an agent-based smart home”. In Proceedings of the International Conference on Smart Homes and Health Telematics (ICOST), 2004.

[69] Das, S. K. and Cook, D. J. (2004b). “Health monitoring in an agent-based smart home by activity predition”. In Proceedings of the International Conference on Smart Homes and Health Telematics, vol. 14, pp.3-14. 2004.

[70] G. Van den Broek, F. Cavallo, L. Odetti, and C. Wehrmann, “Ambient Assisted Living Roadmap,” AALIANCE, Techical Report, 2009.

[71] Tabar, Ali Maleki, Arezou Keshavarz, and Hamid Aghajan. “Smart home care network using sensor fusion and distributed vision-based reasoning.” In Proceedings of the 4th ACM international workshop on Video surveillance and sensor networks, pp.145-154. ACM, 2006.

[72] A. Keshavarz, A. M. Tabar, and H. Aghajan. “Distributed vision-based reasoning for smart home care”. In Proceedings of ACM SenSys Workshop on DSC, October 2006.

[73] Rialle, Vincent, Florence Duchene, Norbert Noury, Lionel Bajolle, and Jacques Demongeot. "Health" smart" home: information technology for patients at home". Telemedicine Journal and E-Health 8, no.4, pp.395-409. 2002.

[74] Zuehlke, D. "SmartFactory –From Vision to Reality in Factory Technologies". In Proceedings of the 17th International Federation of Automatic Control (IFAC) World Congress, Seoul, South Korea, pp. 82-89. 2008.

[75] Zuehlke, Detlef. "SmartFactory - towards a factory-of-things." Annual Reviews in Control 34, no.1, pp.129-138. 2010.

[76] Stokic, Dragan; Kirchhoff, Uwe; Sundmaeker, Harald; "Ambient Intelligence in Manufacturing Industry: Control System Point of View". The 8th IASTED International Conference on Control and Applications; Montreal, Quebec, Canada, pp.63-68. May 2006.

[77] Meixner, Gerrit, Nils Petersen, and Holger Koessling. "User interaction evolution in the SmartFactory KL". In Proceedings of the 24th BCS Interaction Specialist Group Conference, pp.211-220. British Computer Society, 2010.

[78] Finkenzeller, Klaus. RFID Handbook: Radio-frequency identification fundamentals and applications. New York: Wiley, pp.151-158. 1999.

[79] Kubicki, Sébastien, Sophie Lepreux, and Christophe Kolski. "RFID-driven situation awareness on TangiSense, a table interacting with tangible objects." Personal and Ubiquitous Computing 16.8, pp.1079-1094. 2012.

[80] Ota, N. and P. Wright. "Trends in Wireless Sensor Networks for Manufacturing." International Journal of Manufacturing Research 2006 - Vol. 1, No.1, pp.3-17, 2006.

[81] Lucke, Dominik, Carmen Constantinescu, and Engelbert Westkämper. "Smart factory-a step towards the next generation of manufacturing." In Manufacturing Systems and Technologies for the New Frontier, pp.115-118. Springer London, 2008.

[82] Karlsson, Linda. "Distributed generation-the reality of a changing energy market: A market based evaluation and technical description of small wind power and photovoltaics in Sweden." PhD dissertation, Uppsala University, 2011.

[83] S. A. Velastin, B. A. Boghossian, B. P. L. Lo, J. Sun, and M. A. Vicencio-Silva. "PRISMATICA: toward ambient intelligence in public transport environments". IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans, vol.35, no.1, pp.164-182, 2005.

[84] A. Nijholt, "Smart Exposition Rooms: The Ambient Intelligence View". Proc. Electronic Imaging & the Visual Arts (EVA 2004), V. Cappellini & J. Hemsley (eds.), Pitagora Editrice Bologna, pp.100-105. March 2004.

[85] E. Veron & M. Levasseur. "Ethnographie de l'exposition". Paris: Bibliotheque Publique d'Information, Centre Georges Pompidou, 1983.

[86] Flavia Sparacino. "The museum wearable: real–time sensor–driven understanding of visitors' interests for personalized visually-augmented museum experiences." In Museums and the Web, Boston, MA, 2002.

[87] Marti, P., A. Rizzo, L. Petroni, G. Tozzi, and M. Diligenti. "Adapting the museum: a non-intrusive user modeling approach." Courses and Lectures-International Centre for Mechanical Sciences: pp.311-314, 1999.

[88] P Schmidt, Albrecht, Walter Van de Velde, and Gerd Kortuem. "Situated interaction in ubiquitous computing." In CHI'00 extended abstracts on Human factors in computing systems, pp.374-374. ACM, 2000.

[89] Nijholt, A. "Gulliver Project: Performers and Visitors." In: Proceedings EVA 2002 Florence: Electronic Imaging & the Visual Arts, Florence, Italy, pp.241-245. 18-22 March 2002.

[90] Nijholt, Anton. "Smart Exposition Rooms: The Ambient Intelligence View." pp.100-105. 2004.

[91] S-C Chou, W-T Hsieh, F. Gandon and N. Sadeh, "Semantic Web Technologies for Context-Aware Museum Tour Guide Applications", International Workshop on Web and Mobile Information Systems (WAMIS'05), IEEE Computer Society, vol. 2, pp. 709-714. 2005.

[92] Al Takrouri, Bashar, Antonio Canonico, Layda Gongora, Michal Janiszewski, Claudiu Toader, and Andreas Schrader. "Eyejot-a ubiquitous context-aware campus information system." In ICPCA 2007, 2nd International Conference on Pervasive Computing and Applications, pp.122-127. IEEE, 2007.

[93] Sadeh, N., F. Gandon, and O.B. Kwon, "Ambient Intelligence: The MyCampus Experience". In Ambient Intelligence and Pervasive Computing, T.V.a.W. Pedrycz, Editor. ArTech House, 2006.

[94] Bromuri, Stefano, Visara Urovi, and Kostas Stathis. "iCampus: A Connected Campus in the". International Journal of Ambient Computing and Intelligence (IJACI) vol.2, no.1, pp.59-65. 2010.

[95] Youngblood, G. Michael, Diane J. Cook, and Lawrence B. Holder. "The mavhome architecture". Department of Computer Science and Engineering University of Texas at Arlington, Techinal Report, 2004.

[96] Das, Sajal K., and Diane J. Cook. "Designing smart environments: A paradigm based on learning and prediction". In Pattern Recognition and Machine Intelligence, Springer Berlin Heidelberg, pp.80-90. 2005.

[97] Bellman, Wilard F. (2001). "Lighting the Stage: Art and Practice". Third Edition, Chapter 4 –The Control Console, Broadway Press, Inc. 2006, Louisville Kentucky, ISBN 0-911747-40-0.

[98] S. Ondrej, B. Zdenek, F. Petr and H. Ondrej, "Zigbee Technology and Device Design", International Conference on Systems and Mobile Communication, 2006.

[99] Joseph J. Carr John M. Brown - Introduction to Biomedical Equipment Technology, Third Edition - ISBN 0-13-849431-2 - Apr 2, 2010.

[100] G. Pottie and W. Kaiser. "Wireless sensor networks". Communications of the ACM, 43(5):51-58, 2000.

[101] Deshpande, Amol, Carlos Guestrin, and Samuel Madden. "Resource-Aware Wireless Sensor-Actuator Networks". IEEE Data Eng. Bull. Vol.28, no.1, pp.40-47. 2005.

[102] Lee, Myeong-Hyeon, and Yoon-Hwa Choi. "Fault detection of wireless sensor networks". Computer Communications vol.31, no.14, pp.3469-3475. 2008.

[103] J. Chen, S. Kher and A. Somani, "Distributed fault detection of wireless sensor networks". Proceedings of Workshop DIWANS, pp.65-72. 2006.

[104] Surie, Dipak, Olivier Laguionie, and Thomas Pederson. "Wireless sensor networking of everyday objects in a smart home environment." In International Conference on Intelligent Sensors, Sensor Networks and Information Processing. ISSNIP 2008, pp.189-194. IEEE, 2008.

[105] J. Gertler, "Fault Detection and Diagnosis in Engineering Systems". Marcel Dekker, New York, 1998.

[106] Isermann, Rolf. "Model base fault detection and diagnosis methods". In American Control Conference, 1995. Proceedings of the, vol.3, pp.1605-1609. IEEE, 1995.

[107] Isermann, R. "Fault-diagnosis systems: An introduction from fault detection to fault tolerance". Berlin, Germany: Springer. 2006.

[108] L. G. Roberts, "Beyond Moore's law: Internet growth trends," IEEE Computer, vol. vol.33, no.1, pp.117-119, January 2000.

[109] R.O. Duda, P.E. Hart, and D.G. Stork, "Pattern Classification" New York: John Wiley & Sons, 2001, pp. xx + 654, ISBN: 0-471-05669-3. J. Classif. 24, 2 (September 2007), 305-307. DOI=10.1007/s00357-007-0015-9 http://dx.doi.org/10.1007/s00357-007-0015-9.

[110] R. Hanson, J. Stutz, and P. Cheeseman. "Bayesian classification theory". Technical Report FIA-90-12-7-01, NASA Ames Research Center, 1990.

[111] Hänsler, E. "Statistische Signale: Grundlagen und Anwendungen (Statistical Signals - Principles and Applications)" Springer Verlag. 2001.

[112] Jaynes, Edwin T., "Probability Theory: The Logic of Science". Cambridge University Press, pp.592-593. 2003.

[113] Füssel, D., "Fault diagnosis with tree-structured neuro-fuzzy systems", volume Fortscher.-Ber. VDI Reihe 8, 957. VDI Verlag, Düsseldorf, 2002.

[114] K. Fukunaga. "Introduction to statistical pattern recognition". Academic press, 1990.

[115] D. F. Specht, "Generation of polynomial discriminant functions for pattern recognition" IEEE Transactions on Electronic Computers, vol. EC-16, pp.308-319, June 1967.

[116] Schürmann, Jürgen. "Pattern classification: a unified view of statistical and neural approaches". John Wiley & Sons, Inc., 1996.

[117] Nelles, O. "Nonlinear system identification with local linear neuro-fuzzy models". Shaker, Aachen, 1999.

[118] Press, W., Flannery, B., Teukolsky, W., and Vetterling, S. "Neumerical recipes in C". Cambridge University Press, Cambridge, 1988.

[119] J. R. Quinlan. "Induction of decision trees". Machine Learning, vol.1, no.1, pp.81-106. 1986.

[120] J. R. Quinlan. "Simplifying decision trees". International Journal of Man-Machine Studies, vol.27, no.3, pp.221-234. September 1987.

[121] Hart, P. "The condensed nearest neighbor rule". IEEE Transactions on Information Theory, IT-14, pp.515-516. 1968.

[122] Zhang, Guoqiang Peter. "Neural networks for classification: a survey". IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, vol.30, no.4, pp.451-462. 2000.

[123] Isermann, R. and Ballé, P. "Trends in the application of model-based fault detection and diagnosis in technical process". In 13th IFAC World Congress, volume N, pp.1-12, San Francisco, CA, USA, 1996.

[124] Kuncheva L.I., "Fuzzy Classifier Design". Springer-Verlag, Heidelberg, May 2000.

[125] Schneeweiss, Winfrid G. "Reliability theory for large linear systems with helping neighbors". IEEE Transactions on Reliability, vol.41, no.3, pp.343-351. 1992.

[126] Barlow, R and Proschan, F. "Statistical theory of reliability and life testing". Holt, Rinehart & Winston, Inc., 1975.

[127] Freyermuth, Bernd. "An approach to model based fault diagnosis of industrial robots." In Proceedings of IEEE International Conference on Robotics and Automation, pp.1350-1356. 1991.

[128] Isermann, R. "On fuzzy logic applications for automatic control, supervision and fault diagnosis". In proceedings of the 3rd European Congress on Fuzzy and Intelligent Technologies (EUFIT), vol.2, pp.738-753, Aachen, Germany, 1995.

[129] Isermann, R. and Füssel, D. "Supervision, fault detection and fault-diagnosis methods – advanced methods and applications". In Zimmermann, H. –J., editor, Practical applications of fuzzy technologies, pp.119-159. Kluwer Academic, Boston, 1999.

[130] Isermann, Rolf, and Mihaela Ulieru. "Integrated fault detection and diagnosis". In Proceedings of IEEE International Conference on 'Systems, Man and Cybernetics, Systems Engineering in the Service of Humans', pp.743-748., 1993.

[131] Shortliffe, E. "Computer-based medical consultations, MYCIN". Volume 2 of Artificial Intelligence Series. Elsevier, Amsterdam, 1976.

[132] Sanchez, E. "Solutions in composite fuzzy relation equation – application to medical diagnosis in browerian logic". In Gupta, M., Saridis, G., and Gaines, B., editors, Fuzzy automata and decision processes, pp.221-234, North-Holland, Amsterdam, 1977.

[133] Freyermuth, B. Knowledge-based incipient fault diagnosis of industrial robots. In Prepr. IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes (SAFEPROCESS), Pergamon Press, volume 2, pp.31-37, Baden-Baden, Germany, September 1991.

[134] Lee, Wen-Shing, D. L. Grosh, Frank A. Tillman, and Chang H. Lie. "Fault Tree Analysis, Methods, and Applications - A Review". IEEE Transactions on Reliability, vol.34, no.3, pp.194-203, 1985.

[135] Cayrac, D., Dubois, D., and Prade, H. "Handling uncertainty with possibility theory and fuzzy sets in a satellite fault-diagnosis application". IEEE Transactions on fuzzy Systems, vol.4, no.3, pp.251-269, 1996.

[136] Fink, P. and Lusth, J. "Expert systems and diagnostic expertise in the mechanical and electrical domains". IEEE Transactions on Systems, Mann and Cybernetics, vol.17, no.3, pp.340-349, 1987.

[137] Normung, Dindinsf. "Fehlerbaumanalyse-Methode und Bildzeichen (Fault Tree Analysis - Method and Symbols)". Berlin: Beuth 1981.

[138] Pearl, J. "Probabilistic reasoning in intelligent systems: networks of plausible inference". Morgan Kauffmann Publishers, 1988.

[139] Milne, R. "Strategies for diagnosis". IEEE Transactions on Systems, Men & Cybernetics, vol.17, no.3, pp.333-339. 1987.

[140] Reiter, R. "A theory of diagnosis from first principles". Artificial Intelligence, vol.32, pp.57-95, 1987.

[141] Kleer, J. de. "An assumption-based TMS". Artificial intelligence, vol.28, pp.127-162. 1986.

[142] Augusto, J.C., Mccullagh, P., Mcclelland, V., Walkden, J.A. "Enhanced healthcare provision through assisted decision-making in a smart home environment". In Proceedings of the 2nd Workshop on Artificial Intelligence Techniques for Ambient Intelligence - AITAmI'07, IJCAI, pp.27-32. 2007.

[143] Maria Strimpakou, Ioanna Roussaki, Carsten Pils, Michael Angermann, Patrick Robertson, and Miltiades E. Anagnostou. "Context modelling and management in ambient-aware pervasive environments". In Thomas Strang and Claudia Linnhoff-Popien, editors, LoCA, volume 3479 of Lecture Notes in Computer Science, pp.2-15. Springer, 2005.

[144] Sama, Michele, David S. Rosenblum, Zhimin Wang, and Sebastian Elbaum. "Model-based fault detection in context-aware adaptive applications". In Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering, pp.261-271. ACM, 2008.

[145] C. Xu and S. C. Cheung. "Inconsistency detection and resolution for context-aware middleware support". In Proceedings of joint 10th European Software Engineering Conference and 13th ACM SIGSOFT Symposium on the Foundations of Software Engineering, vol.30, no.5, pp.336-345, September 2005.

[146] C. Xu, S. C. Cheung, and W. K. Chan. "Incremental consistency checking for pervasive context". In Proceedings of International Conference on Software Engineering, pp.292–301, May 2006.

[147] Salber, Daniel, Anind K. Dey, and Gregory D. Abowd. "The context toolkit: aiding the development of context-enabled applications". In Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit, pp.434-441. ACM, 1999.

[148] J. De Sousa and D. Garlan, "Aura: An Architectural Framework for User Mobility in Ubiquitous Computing Environments". Proceedings of IEEE-IFIP Conference on Software Architecture, 2002.

[149] Chen, Guanling, and David Kotz. "Context aggregation and dissemination in ubiquitous computing systems". In Proceedings Fourth IEEE Workshop on Mobile Computing Systems and Applications, pp.105-114. IEEE, 2002.

[150] Hong, Jason I., and James A. Landay. "An architecture for privacy-sensitive ubiquitous computing". In Proceedings of the 2nd international conference on Mobile systems, applications, and services, MobiSys. pp.177-189. ACM, 2004.

[151] M.Roman, C.Hess, R.Cerqueira, A.Ranganathan, R.H.Campbell and K.Nahrstedt, "Gaia: A Middleware platform for active spaces". ACM SIGMOBILE Mobile Computing and Communications Review, vol.6, no.4, pp.65-67, 2002.

[152] Chetan, S., Ranganathan, A., & Campbell, R. "Towards fault tolerant pervasive computing". IEEE Technology and Society Magazine, vol.24, no.1, pp.38-44. 2005.

[153] D. Kulkarni and A. Tripathi. "A framework for programming robust context-aware applications". IEEE Transactions on Software Engineering, vol.36, pp.184-197. 2010.

[154] X. Koutsoukos, F. Zhao, H. Haussecker, J. Reich, and P. Cheung, "Fault modeling for fault monitoring and diagnosis of sensor-rich hybrid systems". In Proceedings of IEEE Orlando, FL, pp.793-801. 2001.

[155] Rish, Irina, Mark Brodie, and Sheng Ma. "Efficient fault diagnosis using probing". In AAAI Spring Symposium on Information Refinement and Revision for Decision Making. 2002.

[156] Debouk, Rami, Stéphane Lafortune, and Demosthenis Teneketzis. "Coordinated decentralized protocols for failure diagnosis of discrete event systems". Discrete Event Dynamic Systems vol.10, no.1-2, pp.33-86. 2000.

[157] J. Kurien, X. Koutsoukos, and F. Zhao, "Distributed diagnosis of networked embedded systems". In Proceedings of the 13th International Workshop on Principles of Diagnosis, Semmering, Austria, pp.179-188. 2002.

[158] Darwiche, A. "Model-based diagnosis under real-world constraints". AI Magazine. Vol.21, no.2, pp.57-73. 2000.

[159] Hamscher, W., Console, L. and de Kleer, J. "Readings in Model-Based Diagnosis". Morgan Kaufmann Publishers, Inc., San Mateo, California. 1992.

[160] Embley, David W., and Bernhard Thalheim, eds. "Handbook of conceptual modeling: theory, practice, and research challenges". Springer ISBN 978-3-642-15864-3, 2011.

[161] Satish Mishra (1997). "Visual Modeling & Unified Modeling Language (UML) : Introduction to UML". Rational Software Corporation. Accessed 9 November 2008.

[162] Chen, P. "The entity-relationship model - Toward a unified view of data". ACM Transactions on Database Systems vol.1, no.1, pp.9-36. March 1976.

[163] Bender, E.A. "An Introduction to Mathematical Modeling". New York: Dover Publications. ISBN 0-486-41180-X, 2000.

[164] Temam, Roger. "Infinite dimensional dynamical systems in mechanics and physics". Vol.68. Springer, 1997.
[165] Bickel, Peter J.; Doksum, Kjell A. (2001). Mathematical statistics: Basic and selected topics. 1 (Second (updated printing 2007) ed.). Pearson Prentice-Hall. ISBN 0-13-850363-X. MR 443141.
[166] V. I. Arnold, Ordinary Differential Equations, The MIT Press, ISBN 0-262-51018-9. 1978.
[167] E. L. Ince, "Ordinary Differential Equations". Dover Publications, 1956.
[168] Mark Maier and Eberhardt Rechtin "The Art of Systems Architecture". 2nd edition, 2002.
[169] SEVOCAB: Software and Systems Engineering Vocabulary. Term: block diagram
[170] Mosterman, P. J. and H. Vangheluwe, "Computer automated multi-paradigm modeling: An introduction". Simulation Transactions of the Society for Modeling and Simulation International vol.80, pp. 433–450, special Issue: Grand Challenges for Modeling and Simulation. 2004.
[171] C. Hardebolle and F. Boulanger, "Exploring multi-paradigm modeling techniques". Simulation Transactions of The Society for Modeling and Simulation International, vol.85, pp.688-708, November 2009.
[172] Frédéric Boulanger, Cécile Hardebolle, Christophe Jacquet, Dominique Marcadet. "Semantic Adaptation for Models of Computation". Proceedings of ACSD 2011 (Application of Concurrency to System Design), pp.153-162. IEEE Computer Society. June 2011.
[173] E. A. Lee and D. G. Messerschmitt, "Synchronous data flow". In Proceedings of the IEEE, vol.75, no.9, pp. 1235-1245. September 1987.
[174] J. Eker, J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong, "Taming heterogeneity – the Ptolemy approach," Proceedings of the IEEE, Special Issue on Modeling and Design of Embedded Software, vol.91, no.1, pp.127-144, January 2003.
[175] Balarin, Felice, Luciano Lavagno, Claudio Passerone, Alberto Sangiovanni-Vincentelli, Marco Sgroi, and Yosinori Watanabe. "Modeling and designing heterogeneous systems". In Concurrency and Hardware Design, pp.228-273. Springer Berlin Heidelberg, 2002.
[176] C. G. Cassandras, M. I. Clune, and P. J. Mosterman, "Hybrid system simulation with SimEvents". In Proceedings of the 2nd IFAC Conf. on Analysis and Design of Hybrid Systems, pp.267-269. 2006.
[177] M. Born and E. Wolf. "Principles of Optics: Electromagnetic Theory of Propagation, Interference and Diffraction of Light". Cambridge University Press, Cambridge, England, 7th edition. 2002.
[178] The International Commission on Illumination, Commission Internationale de L'Eclairage. International Lighting Vocabulary, 4th ed. Joint Publication with IEC, CIE 17.4. Vienna 1987
[179] Michael Bass, Handbook of Optics Volume II - Devices, Measurements and Properties, 2nd Edition, McGraw-Hill, ISBN 978-0-07-047974-6 pages 24-40 through 24-47. 1995.
[180] Sillion, Francois X., and Claude Puech. "Radiosity and global illumination". Vol.11. San Francisco: Morgan Kaufmann, 1994.
[181] Van Wylen, Gordon John, Richard Edwin Sonntag, and Claus Borgnakke. "Fundamentals of classical thermodynamics". New York, NY: Wiley, 1994.
[182] Kittel, C. Kroemer, H. "Thermal Physics". Second edition, W.H. Freeman, San Francisco, ISBN 0-7167-1088-9, p.227. 1980.

[183] Chang Lee, Joon. "Thermal Physics – Entropy and Free Energies". World Scientific. ISBN 981-02-4874-1. 2001.

[184] Merayo, Mercedes G., Manuel Núñez, and Ismael Rodríguez. "Formal testing from timed finite state machines". Computer networks vol.52, no.2, pp.432-460. 2008.

[185] E. Chang, A. Pnueli, Z. Manna, "Compositional Verification of Real-Time Systems". Proceedings of the 9th IEEE Symposium on Logic in Computer Science, pp. 458-465. 1994.

[186] J. Ouaknine and J. Worrell, "Some Recent Results in Metric Temporal Logic". In Proc. FORMATS, pp.1-13. 2008.

[187] CBDP project description. http://projects.celtic-initiative.org/cbdp/ - Accessed on 11th June 2013.

[188] OSGi Alliance, OSGi service platform, release 3. IOS Press, Inc., 2003

[189] G. Antoniou and F. van Harmelen, "Web ontology language: OWL," in Handbook on ontologies, S. Staab and R. Studer, Eds. Springer, pp.91-110. 2009.

[190] http://www.w3.org/TR/PR-rdf-syntax/ "Resource Description Framework (RDF) Model and Syntax Specification".

[191] D. Bonino and F. Corno, "DogOnt–ontology modeling for intelligent domotic environments," The Semantic Web – International Semantic Web Conference. ISWC, pp.790-803, 2008.

[192] Jerry R. Hobbs and Feng Pan. (2006) Time Ontology in OWL. http://www.w3.org/TR/owl-time/ Accessed on 11th June 2013.

[193] J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson, "Jena: implementing the semantic web recommendations". In Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters. ACM, pp.74-83. 2004.

[194] A. Chandra and D. Harel, "Horn clause queries and generalizations," The Journal of Logic Programming, vol.2, no.1, pp.1-15, 1985.

[195] Jacquet, Christophe, Ahmed Mohamed*, and Yacine Bellik. "An Ambient Assisted Living Framework with Automatic Self-Diagnosis". International Journal on Advances in Life Sciences vol.5, no.1. 2013.

[196] Boulanger, Frédéric, and Cécile Hardebolle. "Simulation of Multi-Formalism Models with ModHel'X". In 1st International Conference on Software Testing, Verification, and Validation, pp.318-327. IEEE, 2008.

[197] Romuald Deshayes, Christophe Jacquet, Cécile Hardebolle, Frédéric Boulanger, Tom Mens. "Heterogeneous Modeling of Gesture-Based 3D Applications". Proceedings of MPM 2012 (Multi-Paradigm Modeling workshop at Models 2012), pp.1-6. October 2012.

[198] Tamara A. Papalias and Mike Wong, "Making Sense of Light Sensors". Application notes, CA: Intersil Americas Inc. 2007.

[199] R. Sharifi and R. Langari, "Sensor Fault Diagnosis with a Probabilistic Decision Process". Mechanical System and Signal Processing, Vol.34, No.1-2, pp.146-155, January 2013.

[200] Lampis, M., and J. D. Andrews. "Bayesian belief networks for system fault diagnostics". Quality and Reliability Engineering International 25.4 (2009): 409-426.

[201] V. Venkatasubramanian, R. Rengaswamy, K. Yin, S.N. Kavuri "A review of process fault detection and diagnosis Part I: Quantitative model-based methods" Computer and Chemical Engineering, vol.27, pp.293-311. 2003.

[202] S.N. Maheshwari, S.L. Hakimi "On models for diagnosable systems and probabilistic fault diagnosis" IEEE Transactions on Computers, vol.100, no.3, pp.228-236. 1976.

[203] P.H. Ibargüengoytia, S. Vadera, L.E. Sucar "A probabilistic model for information and sensor validation" Computers Journal, vol.49, no.1, pp.113-126. 2006.

[204] Smets, Philippe, and Robert Kennes. "The transferable belief model." Artificial intelligence vol.66. no.2, pp.191-234. 1994.

[205] Yager, Ronald R. "On the Dempster-Shafer framework and new combination rules." Information sciences vol.41, no.2, pp.93-137. 1987.

[206] Ph. Smets. The application of the transferable belief model to diagnostic problems. International Journal of Intelligent Systems, vol.13, pp.127-157. 1998.

[207] Rakar, Andrej, Đani Juričić, and Peter Ballé. "Transferable belief model in fault diagnosis." Engineering Applications of Artificial Intelligence vol.12, no.5, pp. 555-567. 1999.

[208] WANG, Hongm, Tian-You CHAI, Jin-Liang DING, and Martin BROWN. "Data driven fault diagnosis and fault tolerant control: some advances and possible new directions." Acta Automatica Sinica vol.35, no.6, pp.739-747. 2009.

[209] "http://www.w3.org/TR/sparql11-overview/". w3.org. 21/03/2013. Accessed 26th June 2013.

[210] Bernaras, A., I. Laresgoiti, N. Bartolome, and J. Corera. "An ontology for fault diagnosis in electrical networks." In Proceedings of ISAP'96., IEEE International Conference on Intelligent Systems Applications to Power Systems, pp.199-203., 1996.

[211] Gharsellaoui, Asma, Yacine Bellik, and Christophe Jacquet. "Requirements of Task Modeling in Ambient Intelligent Environment". In Ambient 2012, The Second International Conference on Ambient Computing, Applications, Services and Technologies, pp.71-78. 2012.

[212] Steggles, Pete, and Stephan Gschwind. "The Ubisense smart space platform." In Adjunct Proceedings of the Third International Conference on Pervasive Computing, vol.191, pp.73-76. 2005.

[213] Tsao, Jeff Y. "Solid-state lighting: lamps, chips, and materials for tomorrow". Circuits and Devices Magazine, IEEE vol.20, no.3, pp.28-37. 2004.

[214] Schlyter, Paul. "Radiometry and photometry in astronomy FAQ." Stockholm: Paul Schlyter Home Page. http://stjarnhimlen.se/comp/radfaq.html (2006). Accessed 5th November 2013.

[215] Wark, Kenneth and Richards, Donald E., Thermodynamics, 6th Ed., McGraw-Hill, 1999.

[216] Jacquet, Christophe, Ahmed Mohamed*, Frédéric Boulanger, Cécile Hardebolle, and Yacine Bellik. "Building heterogeneous models at runtime to detect faults in ambient-intelligent environments." In Proceedings of the 8th Workshop on Models at Run.time, vol.1079, pp.52-63. 2013.