Problemspezifikation eines Systems zur Steuerung des Lichts und der Temperatur in einem Stockwerk eines Gebäudes der Universität Kaiserslautern

(Praktikumsversion 1.0)

Martin Kronenburg
AG Effiziente Algorithmen

Christian Peper
AG Rechnernetze

Universität Kaiserslautern

9th April 1999

Contents

Introduction

Description Classes

Einleitung

Dieses Dokument enthält eine *Problemspezifikation* für die Steuerung des Lichts und der Temperatur in dem vierten Stockwerk des Gebäudes 32 der Universität Kaiserslautern. Eine Problemspezifikation beschreibt ein bestimmtes System, das entwickelt werden soll. Dabei hat ein Problemspezifikation eine Brückenfunktion zwischen dem Kunden auf der einen Seite und dem Entwickler auf der anderen Seite. Im allgemeinen sind mehrere Gespräche zwischen Kunden und Entwickler erforderlich, bis beide eine möglichst gleiche Vorstellung von dem System haben, das entwickelt werden soll.

Letztlich beinhaltet eine Problemspezifikation sowohl eine Beschreibung des Ist–Zustands, zum Beispiel die bereits vorhandene Installation oder physikalische Gesetzmäßigkeiten, als auch eine Beschreibung des Soll–Zustands, zum Beispiel, daß die Temperatur unter bestimmten Voraussetzungen stets oberhalb einer bestimmten Grenze zu liegen hat. Auf diese Zweiteilung wird später noch genauer eingegangen.

Die vorliegende Problemspezifikation ist nicht so geschrieben, daß man sie von vorne nach hinten als Ganzes durchlesen sollte; sie ist vielmehr in Form eines Nachschlagewerkes aufgebaut. Um in der Problemspezifikation bestimmte Teile einfacher und schneller finden zu können, ist sie strukturiert und mit einigen Navigationshilfen ausgestattet. Dies wird in den folgenden Abschnitten erläutert.

Im Gegensatz zu diesen Erläuterungen ist die eigentliche Problemspezifikation in Englisch verfaßt. Daher wird bei den folgenden Erklärungen stets auch der englische Begriff, wie er in der Problemspezifikation verwendet wird, angegeben. Ebenso sind alle Beispiele in Englisch.

Problemspezifikation

Global Signature
⋮

Description Class <name$_1$ >
⋮

⋮

Description Class <name$_n$ >
⋮

Main Description Class <name>
⋮

Abbildung 1: Aufbau einer Problemspezifikation

1 Aufbau einer Problemspezifikation

Eine Problemspezifikation enthält eine *globale Signatur* (*Global Signature*), die am Anfang des Dokuments steht, sowie eine Menge von *Beschreibungsklassen* (*Description Classes*). Eine dieser Beschreibungsklassen ist als *Hauptbeschreibungsklasse* (*Main Description Classes*) ausgezeichnet. Sie befindet sich am Ende des Dokuments (siehe Abbildung 1).

Im folgenden wird nun zunächst der Aufbau einer Beschreibungsklasse näher erläutert. Die globale Signatur sowie die Hauptbeschreibungsklasse werden anschließend kurz als Spezialfälle einer Beschreibungsklasse erklärt.

1.1 Beschreibungsklassen (Description Classes)

In einer Beschreibungsklasse wird ein bestimmter Teil des zu entwickelnden Systems beschrieben, z.B. ein Sensor, die Lichtsteuerung auf einem Flur oder die Temperatursteuerung in einem Raum. Eine Beschreibungsklasse besitzt einen eindeutigen Namen und ist aufgeteilt in sechs Abschnitte. Siehe Abbildung 2.

Description Class <name>
⋮
INTENTION
⋮
FORMAL PARAMETERS
⋮
BASE CLASSES
⋮
SIGNATURE
⋮
DOMAIN KNOWLEDGE
⋮
REQUIREMENTS/SPECIFICATIONS
⋮

Abbildung 2: Aufbau einer Beschreibungsklasse

Außer dem INTENTION–Abschnitt kann jeder andere Abschnitt fehlen. Mindestens einer der anderen fünf Abschnitte ist jedoch immer vorhanden. Diese sechs Abschnitte einer Beschreibungsklasse werden im folgenden erläutert. Aus Verständnisgründen werden die beiden Abschnitte FORMAL PARAMETERS und BASE CLASSES zuletzt betrachtet.

1.1.1 Abschnitt INTENTION

Dieser Abschnitt ist in jeder Beschreibungsklasse enthalten. Hier wird kurz und informell beschrieben, welcher Teil des Systems in dieser Beschreibungsklasse betrachtet wird und welche Besonderheiten zu beachten sind.

1.1.2 Abschnitt SIGNATURE

In diesem Abschnitt werden die *Bezeichner* eingeführt, die in der Beschreibungsklasse verwendet werden, um über den betrachteten Teil des Systems reden zu können. Als Beispiel nehme man eine Beschreibungsklasse, die die Steuerung der Temperatur in einem Raum beschreibt. In einer solchen Beschreibungsklasse möchte man beispielsweise über die aktuelle Raumtemperatur reden oder über die Stellung des Heizkörperventils oder darüber, ob sich eine Person im Raum befindet. Um über solche *Phänomene*, die in der realen Welt vorkommen, reden zu können, werden Bezeichner benötigt.

Je nachdem wie man über ein bestimmtes Phänomen reden möchte, wird ein Bezeichner der folgenden sieben Arten verwendet.

- Domäne (Domain)
- Funktion (Function)
- zeitabhängige Funktion (Timed Function)
- Prädikat (Predicate)
- zeitabhängiges Prädikat (Timed Predicate)
- Zeitkonstante (Time Constant)
- Objekt (Object)

Abbildung 3: Arten von Bezeichnern

Bei der Vorstellung der einzelnen Bezeichnerarten wird im folgenden zunächst die Art des Phänomens erläutert, das mit einem entsprechenden Bezeichner repräsentiert werden kann. Anschließend wird an Hand eines Beispiels angegeben, wie ein solcher Bezeichner eingeführt wird. Zur Einführung eines Bezeichners wird bei allen Arten ein tabellenartiger Aufbau verwendet.

1.1.3 Domänen (Domains)

In einer *Domäne* ist eine Menge von Werten zusammengefaßt. Ein Beispiel für eine

Domäne ist die Menge der Werte, die ein Temperatursensor liefern kann. Um über dieses Phänomen, d.h. über diese Menge, in eindeutiger Weise reden zu können, wird der Bezeichner *TEMP* eingeführt. Dies geschieht wie folgt:

Domain $TEMP = \{-40 \ldots 70\} <$ INTEGER
Intention : This domain contains the possible temperature values in degrees Celsius that can be returned by a temperature sensor.

Beispiel: Domäne

Der Eintrag bei *Intention* gibt an, welches Phänomen in der realen Welt durch diesen Bezeichner repräsentiert wird.

Der formale Teil der Definition gibt die Menge der möglichen Werte an, hier -40 bis 70. Zusätzlich kann durch das Symbol $<$ angegeben werden, ob diese Domäne eine Unterdomäne einer anderen bereits eingeführten Domäne ist. In dem obigen Beispiel ist *TEMP* eine Teilmenge der ganzen Zahlen. Damit wird auch ausgedrückt, daß der Temperatursensor zum Beispiel nicht den Wert 4.11, sondern stattdessen 4 oder 5 zurückliefert.

Folgende Domänen werden nicht explizit eingeführt, können aber trotzdem verwendet werden:

NAT Dieser Bezeichner repräsentiert die Menge der natürlichen Zahlen.

INTEGER Dieser Bezeichner repräsentiert die Menge der ganzen Zahlen.

REAL Dieser Bezeichner repräsentiert die Menge der reellen Zahlen.

TIME Dieser Bezeichner repräsentiert die Menge aller möglichen Zeitpunkte. Diese Menge entspricht der Menge der nicht–negativen reellen Zahlen.

STRING Dieser Bezeichner repräsentiert die Menge aller endlichen Zeichenketten über dem Alphabet $\{a, \ldots, z, A, \ldots, Z, , 0, \ldots 9\}$.

Die Bezeichner von Domänen sind immer in Großbuchstaben geschrieben.

1.1.4 Funktionen (Functions), zeitabhängige Funktionen (Timed Functions)

Eine Funktion ordnet den Werten einer Menge Werte einer anderen Menge zu. Beispielsweise kann es eine Funktion *tempVal* geben, die beschreibt, welchen Wert ein Temperatursensor zu einem bestimmten Zeitpunkt liefert. Dies bedeutet, daß die Funktion *tempVal* jedem Zeitpunkt einen Wert aus der Domäne *TEMP* zuordnet.

Die meisten Funktionen, die in dieser Problemspezifikation verwendet werden, beschreiben Phänomene, die sich im Laufe der Zeit verändern können, wie zum Beispiel der

Wert, den ein Temperatursensor liefert. Um nicht jedesmal explizit die Domäne *TIME* angeben zu müssen, die die Menge der Zeitpunkte repräsentiert, werden solche Funktionen als *zeitabhängige Funktionen* (*Timed Functions*) bezeichnet. Im folgenden Beispiel steht *Timed Function tempVal* () → *TEMP* also als Abkürzung für *Function tempVal* (*TIME*) → *TEMP*.

Timed Function *tempVal*() → *TEMP*	
Intention	: This function represents the value returned by a temperature sensor.
Scope	: environment controlled, visible for the machine

Beispiel: Zeitabhängige Funktion

Der Eintrag bei *Intention* gibt an, welches Phänomen in der realen Welt durch diesen Bezeichner repräsentiert wird.

Um den Eintrag bei *Scope* zu verstehen, sind einige Vorbemerkungen erforderlich. Bei der Entwicklung eines Systems gehen wir davon aus, daß man dieses System in zwei Komponenten aufteilen kann: die *Umgebung* (*environment*) und die *Maschine* (*machine*). Die Umgebung ist die Komponente des zu entwickelnden Systems, die bereits vorhanden ist. Dazu gehören zum Beispiel Sensoren oder Aktuatoren, die bereits installiert sind, aber auch physikalische Phänomene wie zum Beispiel die Temperatur in einem Raum oder die Sonneneinstrahlung durch ein Fenster in einen Raum.

Die Maschine ist die Komponente des zu entwickelnden Systems, die noch nicht vorhanden ist. Hierzu gehören zum Beispiel Algorithmen, die in Abhängigkeit von der Außentemperatur und der Anwesenheit einer Person in einem Raum die Einstellung der Heizungsventile berechnen.

Wir gehen davon aus, daß man für jedes Phänomen genau eine dieser beiden Komponenten auszeichnen kann, die dieses Phänomen *kontrolliert*. Kontrolle bedeutet dabei, daß diese Komponente des Systems das Ändern oder auch das Nicht–Ändern des Phänomens kontrolliert. Für die kontrollierende Komponente des Systems ist das Phänomen auch *sichtbar*. Für die andere Komponente, die das Phänomen nicht kontrolliert, kann es auch sichtbar sein, muß es jedoch nicht.

Beispielsweise wird die Temperatur in einem Raum von der Umgebung kontrolliert und ist für die Maschine nicht sichtbar. Ist ein Temperatursensor in dem Raum bereits installiert, so wird der Wert, den dieser Sensor liefert, auch von der Umgebung kontrolliert, ist aber sichtbar für die Maschine. Ein in dem Raum bereits installiertes Heizungsventil wird hingegen i.a. von der Maschine kontrolliert und ist sichtbar für die Umgebung. Zu Phänomenen, die von der Maschine kontrolliert und nicht sichtbar für die Umgebung sind, gehören zum Beispiel Werte, die die Maschine berechnet. Solche Phänomene dürfen in einer Problemspezifikation nur vorkommen, wenn sie vom Kunden, der den Auftrag für die Entwicklung des Systems gegeben hat, explizit eingeführt werden.

Aufgrund dieser Klassifikation (Kontrolle und Sichtbarkeit) wird jeder Funktion (und auch jedem Prädikat, siehe Abschnitt 1.1.6) einer der folgenden vier Wirkungsbereiche

(scopes) zugeordnet:

- environment controlled, not visible for the machine
- environment controlled, visible for the machine
- machine controlled, not visible for the environment
- machine controlled, visible for the environment

1.1.5 Zeitkonstanten (Time Constants)

Neben den zeitabhängigen Funktionen kommt in der Problemspezifikation eine weitere Art von Funktionen so häufig vor, daß sie einen eigenen Namen erhält. Dabei handelt es sich um Funktionen, die nicht von der Zeit abhängen, sondern genau einen bestimmten Wert aus der Zeitdomäne festlegen. Eine solche Funktion heißt *Zeitkonstante* (*Time Constant*). Wenn man zum Beispiel ausdrücken möchte, daß ein Fenster innerhalb einer Stunde eine bestimmte Zeit *durWinOpen* geöffnet werden soll, so ist *durWinOpen* eine Zeitkonstante.

Time Constant *durWinOpen*	
Intention	: This time constant represents the duration a window has to be open during one hour.
Scope	: environment controlled, not visible for the machine

Beispiel: Zeitkonstante

1.1.6 Prädikate (Predicates), zeitabhängige Prädikate (Timed Predicates)

Häufig ist es ausreichend, von einem Phänomen zu wissen, ob es gilt oder nicht. Von einem Raum will man zum Beispiel nur wissen, ob sich zu einem bestimmten Zeitpunkt eine Person in ihm aufhält oder nicht. Es ist nicht wesentlich, wie viele Personen sich in dem Raum aufhalten. Um solche Ja–Nein–Zustände zu beschreiben, werden *Prädikate* verwendet. Ein Prädikat gilt (ist wahr) oder es gilt nicht (ist falsch).

Wie bei den Funktionen werden in der Problembeschreibung fast ausschließlich Prädikate verwendet, bei denen es von dem betrachteten Zeitpunkt abhängt, ob sie gelten oder nicht. Diese werden analog zu den Funktionen als *zeitabhängige Prädikate* (*Timed Predicates*) bezeichnet.

Timed Predicate *roomEmpty*	
Intention	: This predicate is true if and only if no person is the room.
Scope	: environment controlled, not visible for the machine

Beispiel: Zeitabhängiges Prädikat

Die Bedeutungen der Einträge bei *Intention* und *Scope* entsprichen denen bei Funktionen.

1.1.7 Objekte (Objects)

Aufgrund der Strukturierung in Beschreibungsklassen ist es möglich, eine Instanz einer Beschreibungsklasse in einer anderen Beschreibungsklasse zu aggregieren. Gibt es beispielsweise eine Beschreibungsklasse *MotionDetector*, die einen Bewegungsmelder repräsentiert, so kann man eine oder mehrere Instanzen dieser Beschreibungsklasse in einer Beschreibungsklasse *RoomTemp* für die Temperatursteuerung in einem Raum aggregieren.

Object *md* : *MotionDetector*	
Intention	: This object represents the room motion detector.

Beispiel: Objekt

Die Aggregation eines Objektes bedeutet, daß alle Bezeichner, die in der aggregierten Klasse (im obigen Beispiel in der Beschreibungsklasse *MotionDetector*) eingeführt worden sind, auch in der aggregierenden Beschreibungsklasse (im obigen Beispiel in der Beschreibungsklasse *RoomTemp*) verfügbar sind. Dabei wird diesen Bezeichnern jedoch der Bezeichner des Objektes, durch einen "." getrennt, vorangestellt (im obigen Beispiel *md*). Gibt es also zum Beispiel in der Beschreibungsklasse *MotionDetector* den Bezeichner *measEntity*, so gibt es aufgrund der Aggregation in der aggregierenden Beschreibungsklasse *RoomTemp* den Bezeichner *md.measEntity*. Ein weiterer Aspekt der Aggregation wird am Ende von 1.1.11 erläutert.

Damit sind alle möglichen Arten von Bezeichnern eingeführt.

1.1.8 Abschnitt Domain Knowledge

In diesem Abschnitt einer Beschreibungsklasse wird der *Ist–Zustand* des betrachteten Teils des Systems beschrieben. Es wird also beschrieben, wie sich das System verhält, ohne daß die Maschine vorhanden ist. Zum Beispiel gehört die Aussage, welche Beziehung zwischen dem von einem Sensor gemessenen Wert und dem in der Umwelt tatsächlich vorhandenen Wert vorliegt, zum Domänenwissen. Hier wird es im allgemeinen zu Verzögerungen kommen und zum Beispiel zu Abweichungen aufgrund von Rundungsfehlern. Das folgende Beispiel beschreibt einen solchen Zusammenhang.

Property D_S_1	
Formal	: $\Box(envEntity \rhd^{modifyDuringMeas}_{\leq reactionTime} measEntity)$
NL	: The measured value of a sensor (*measEntity*) is always derived from the value in the real world (*envEntity*) using the function *modifyDuringMeas*. A change of the value in the real world is always propagated within the time *reactionTime*.

Beispiel: Domänenwissen

Jede Aussage, die zum Domänenwissen gehört, erhält einen eindeutigen Namen zugeordnet, der hinter dem Schlüsselwort **Property** steht. Im obigen Beispiel ist dies D_S_1[1]. Jede Aussage wird formal, d.h. als Formel in einer Temporallogik, wiedergegeben (hinter dem Schlüsselwort *Formal*), und in natürlicher Sprache (hinter dem Schlüsselwort *NL*[2]). Eine genauere Erläuterung der temporallogischen Formeln erfolgt hier nicht.

1.1.9 Abschnitt Requirements/Specifications

In diesem Abschnitt einer Beschreibungsklasse wird der *Soll-Zustand* des betrachteten Teils des Systems beschrieben. Es wird also beschrieben, wie sich das System unter Einflußnahme der Maschine verhalten soll. Das folgende Beispiel gibt die Anforderung wieder, daß ein Benutzer darüber informiert werden soll, falls ein Sensor einen Fehler aufweist.

Property P_MS_4	
Formal	: $\Box([malDetMach] \rightarrow \Diamond_{\leq T_{infDelay}} userInformed)$
NL	: Whenever the machine assumes that a malfunction of the sensor has occurred, the user is informed about this within $T_{infDelay}$ milliseconds.

Beispiel: Anforderungen/Spezifikationen

Der tabellarische Aufbau einer solchen formalisierten Anforderung entspricht dem einer Aussage des Domänenwissens.

Eine Teilmenge der Anforderungen, nämlich die, die den Startzustand des Systems festlegen, stehen zu Beginn des Abschnitts Requirements/Specifications. Ein solcher Teilabschnitt wird durch das Schlüsselwort Initialization eingeleitet und durch das Schlüsselwort End of Initialization beendet.

[1] **D** steht für **d**omain knowledge, **S** für sensor

[2] NL steht für *natural language*.

INITIALIZATION
Property I_RL_1
Formal : $\neg manualControl$
NL : Initially the light is not controlled manually.
END OF INITIALIZATION

Beispiel: Anforderung für den Startzustand

Es gibt auch Anforderungen, die sich nicht mit Hilfe der hier verwendeten Temporallogik formalisieren lassen. Dies sind Aussagen, die nicht das Verhalten des zu entwickelnden Systems betreffen. Solche Aussagen werden ebenfalls in der Problemspezifikation im Abschnitt REQUIREMENTS/SPECIFICATIONS angegeben, jedoch mit dem Schlüsselwort *NonFormal*. In solchen Fällen gibt es also nur eine natürlichsprachliche Fassung der Anforderung.

Property P_LCP_1
NonFormal : The control panel should be easy and intuitive to use.

Beispiel: Nichtformalisierte Anforderung

1.1.10 Abschnitt FORMAL PARAMETERS

Um Beschreibungsklassen allgemeiner definieren zu können, ist es möglich, sie mit Parametern zu versehen. In der Problemspezifikation kommen zwei Arten von Parametern vor: *Domänenparameter* und *Anzahlparameter*. Beide Arten werden jeweils an Hand eines Beispiels erläutert.

Verschiedene Arten von Sensoren unterscheiden sich unter anderem in den Werten, die sie messen und zurückliefern, zum Beispiel Temperatur– oder Helligkeitssensoren. Um dennoch eine allgemeine Klasse für Sensoren angeben zu können, in der die Gemeinsamkeiten für alle Sensorenarten nur einmal aufgeführt werden und von der speziellere Arten von Sensoren abgeleitet werden können (siehe Abschnitt 1.1.11), wird eine allgemeine Beschreibungsklasse *Sensor* definiert, bei der die beiden entsprechenden Domänen noch nicht endgültig festgelegt werden. Dies sind im folgenden Beispiel die Domänen *ENV_DOMAIN* und *MEASURED_DOMAIN*.

Description Class Sensor

INTENTION
This class provides all definitions to describe the behavior of a *sensor*. ...

FORMAL PARAMETERS
Domain *ENV_DOMAIN*
Intention : This domain contains the possible values of the measured phenomenon existing in the real world.

Domain *MEASURED_DOMAIN*
Intention : This domain contains the possible measured values that are returned by the sensor.
⋮

Beispiel: Domänenparameter

Wir kommen nun zu dem Beispiel für Anzahlparameter. Im allgemeinen haben nicht alle Räume gleich viele Fenster. Um dennoch Gemeinsamkeiten von Räumen mit einer unterschiedlichen Anzahl von Fenstern in einer Beschreibungsklasse beschreiben zu können, kann die genaue Anzahl der Fenster als Parameter angegeben werden. Im folgenden Beispiel erfolgt dies mit Hilfe des Parameters *numWinSect*, der formal als Funktion definiert wird.

Description Class Room

INTENTION
This class provides all definitions to describe the behavior common to any kind of room. ...

FORMAL PARAMETERS
Function $numWinSect() \rightarrow NAT$
Intention : This function represents the number of window sections that belong to a room.
⋮

Beispiel: Anzahlparameter

1.1.11 Abschnitt BASE CLASSES

Wie bereits mehrfach angedeutet, kann eine Beschreibungsklasse von einer anderen Beschreibungsklasse abgeleitet werden. Als Beispiel betrachte man die Beschreibungsklasse eines binären Sensors. Ein binärer Sensor ist ein Sensor, der nur zwei Werte zurückliefern kann: 0 oder 1. Ein Bewegungsmelder ist ein Beispiel für einen binären Sensor. Das folgende Beispiel zeigt wie die Beschreibungsklasse eines binären Sensors von der eines (allgemeinen) Sensors (siehe das Beispiel für Domänenparameter) abgeleitet wird. Dabei erfolgt eine Instantiierung der Domänenparameter.

Description Class BinarySensor

INTENTION
This class provides all definitions to describe the behavior of a *binary sensor*. A binary sensor is a sensor that can return a 0 or a 1.

BASE CLASSES
Class $Sensor(ENV\_DOMAIN = BINARY,$
$MEASURED\_DOMAIN = BINARY)$

Intention : All definitions of Sensor are also provided here using the values 0 and 1 as the possible values of the phenomenon existing in the real world and using the values 0 and 1 as the possible measured values.

⋮

Beispiel: Vererbung durch Basisklassen

Wird eine Beschreibungsklasse von einer anderen Beschreibungsklasse abgeleitet, so *erbt* sie alle Bezeichner und Aussagen, die in dieser Beschreibungsklasse eingeführt bzw. gemacht wurden. In diesem Beispiel bedeutet dies also, daß alle Bezeichner, die in der Beschreibungsklasse *Sensor* eingeführt wurden, auch in der Beschreibungsklasse *BinarySensor* verwendet werden können. Des weiteren sind auch alle Aussagen, die in den Abschnitten DOMAINKNOWLEDGE und REQUIREMENTS/SPECIFICATIONS der Beschreibungsklasse *Sensor* gemacht wurden, in der Beschreibungsklasse *BinarySensor* gültig.

Analog verhält es sich, wenn ein Objekt als Instanz einer Beschreibungsklasse in einer anderen aggregiert wird. Auch in diesem Fall werden die Aussagen, d.h. das Domänenwissen und die Anforderungen/Spezifikationen, der aggregierten Klasse übernommen. Die in den Aussagen vorkommenden Bezeichner werden dabei, wie bereits in Abschnitt 1.1.7 beschrieben, erweitert.

1.2 Globale Signatur (Global Signature)

Die globale Signatur zu Beginn einer Problemspezifikation ist eine Beschreibungsklasse, die nur aus dem Abschnitt SIGNATUR besteht. Es werden also auch hier Bezeichner eingeführt. Der Unterschied ist jedoch der, daß die Bezeichner, die in der globalen Signatur eingeführt werden, für alle Beschreibungsklassen sichtbar sind, also in allen Beschreibungsklassen verwendet werden können. Daher werden in der globalen Signatur Bezeichner eingeführt, die nicht nur einem bestimmten Teil des Systems zugeordnet werden können. So wird zum Beispiel in verschiedenen Beschreibungsklassen die Domäne *BINARY* benötigt, die die beiden Zahlen 0 und 1 enthält. Diese Domäne wird daher in der globalen Signatur eingeführt.

1.3 Hauptbeschreibungsklasse (Main Description Class)

Die eigentliche Spezifikation des Systems, das entwickelt werden soll, befindet sich in der Hauptbeschreibungsklasse (Main Description Class) am Ende des Dokuments. Alle anderen Beschreibungsklassen sind Hilfsmittel, um diese Hauptbeschreibungsklasse kompakt darzustellen. Hier wird zum Beispiel durch Aggregation entsprechender Objekte angegeben, welche Räume konkret zum System dazugehören. Es ist hier jedoch nicht mehr erforderlich, das Verhalten jedes einzelnen Raumes erneut zu spezifizieren. Dies ist bereits in der Beschreibungsklasse Room für alle Räume geschehen.

2 Navigationshilfen

Um in der Problemspezifikation Informationen leichter finden zu können, enthält sie neben den eigentlichen Informationen, d.h. den Beschreibungsklassen, noch weitere *Navigationshilfen*.

2.1 Inhaltsverzeichnis

Im Inhaltsverzeichnis sind alle Beschreibungsklassen aufgelistet und gruppiert. In einer *Gruppe* werden Beschreibungsklassen nach verschiedenen Kriterien zusammengefaßt. So werden zum Beispiel alle Sensoren und Aktuatoren jeweils zu einer Gruppe zusammengefaßt oder die Beschreibungsklassen, mit deren Hilfe die Anwesenheit einer Person in einem Gebäudebereich definiert wird (Group *Occupancy*). Diese Gruppen dienen ausschließlich der Strukturierung und Übersichtlichkeit; sie haben keine Bedeutung für den Inhalt der Problemspezifikation.

2.2 Index

Am Ende des Dokuments befindet sich ein alphabetisch sortierter Index. In ihm sind alle in der Problemspezifikation verwendeten Bezeichner sowie die Namen der Beschreibungsklassen aufgeführt.

2.3 Aggregations– und Vererbungshierarchie

Vor den Beschreibungsklassen sind die Aggregations– und Vererbungsbeziehungen zwischen den Beschreibungsklassen als Graphen dargestellt. Diese graphische Übersicht besteht aus fünf Abbildungen. Jede Abbildung beinhaltet die Beziehungen innerhalb einer Gruppe (siehe Absatz 2.1). Abbildung 6 ist die zentrale Übersicht. Sie gibt die Beziehungen der Gruppe *Building Unists* wieder.

Ausgehend von der Hauptbeschreibungsklasse *Floor* werden hier die Beziehungen zwischen den Klassen soweit dargestellt, bis eine Klasse nicht mehr feiner strukturiert ist oder bis eine Klasse einer anderen Gruppe erreicht ist. Klassen, die eine feinere Struktur besitzen, werden in einer der nachfolgenden Abbildungen dargestellt.

- Gruppe *Sensoren* (*Sensors*); Abbildung 7
- Gruppe *Aktuatoren* (*Actuators*); Abbildung 8
- Gruppe *Kombinierte Sensosren und Aktuatoren* (*CombinedSensorsandActuators*); Abbildung 9
- Gruppe *Mensch–Maschine–Schnittstelle* (*HumanMachineInterface*); Abbildung 10

Zur Darstellung der Aggregations- und Vererbungsbeziehungen werden die folgenden Symbole verwendet. Werden in einer Beschreibungsklasse A n Objekte der Beschreibungsklasse B aggregiert, so wird dies wie folgt dargestellt:

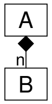

Abbildung 4: Graphische Darstellung der Aggregation

Wird eine Beschreibungsklasse D von einer Beschreibungsklasse C abgeleitet, d.h. D erbt von C, so wird dies wie folgt dargestellt:

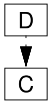

Abbildung 5: Graphische Darstellung der Vererbung

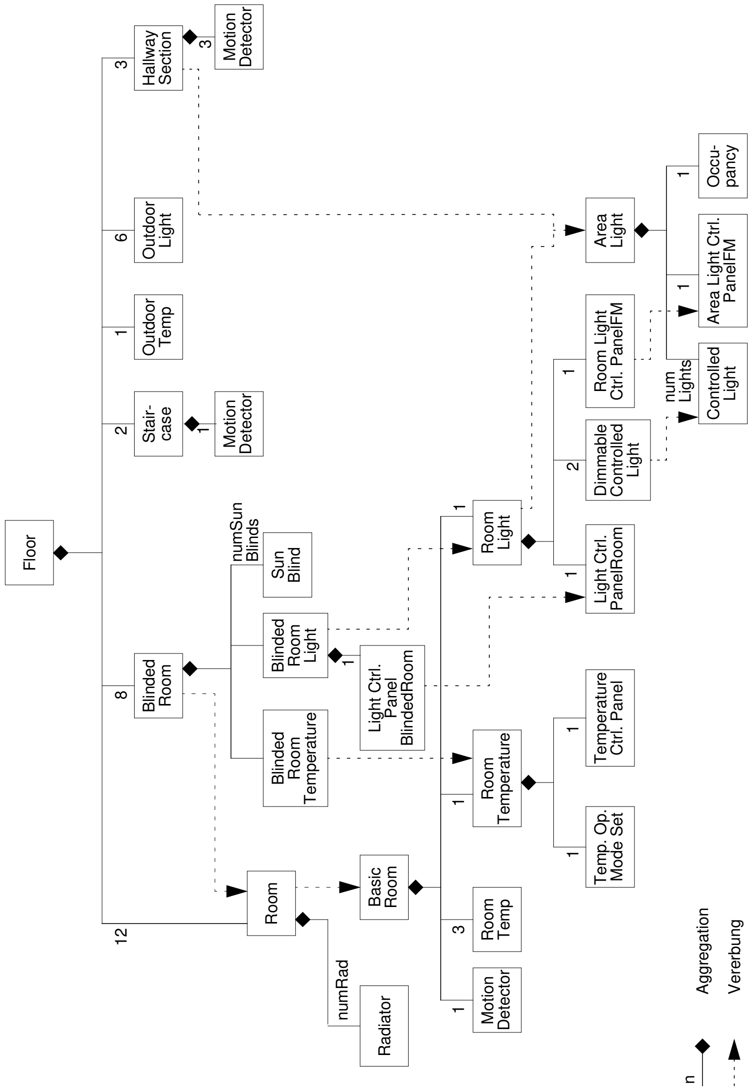

Abbildung 6: Klassenstruktur der Gruppe *BuildingUnits*

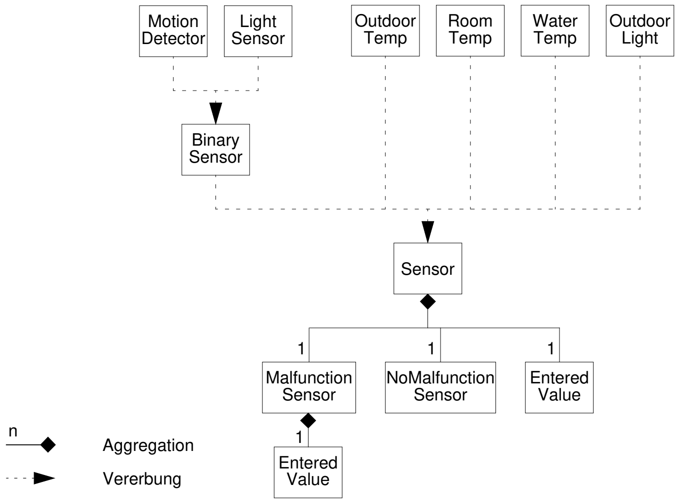

Abbildung 7: Klassenstruktur der Gruppe *Sensors*

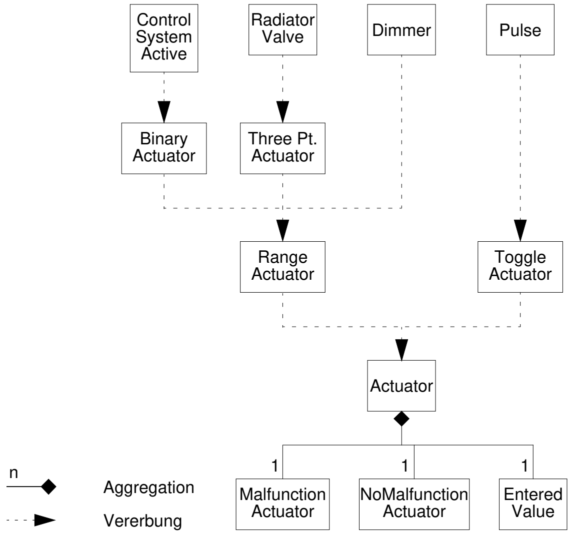

Abbildung 8: Klassenstruktur der Gruppe *Actuators*

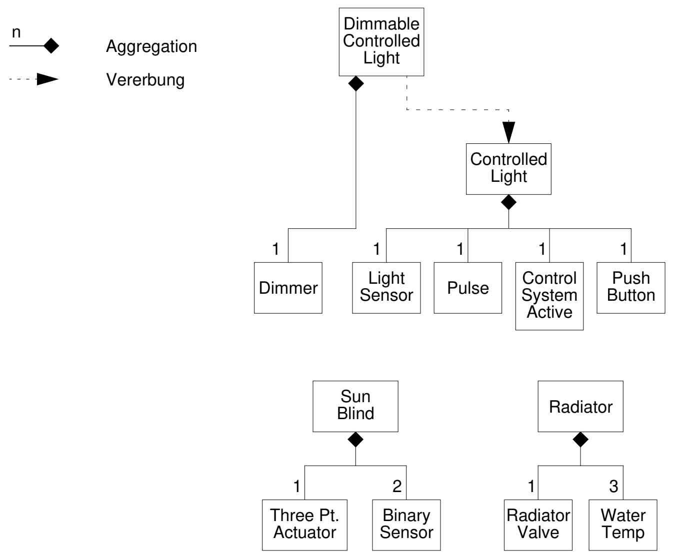

Abbildung 9: Klassenstruktur der Gruppe *CombinedSensorsAndActuators*

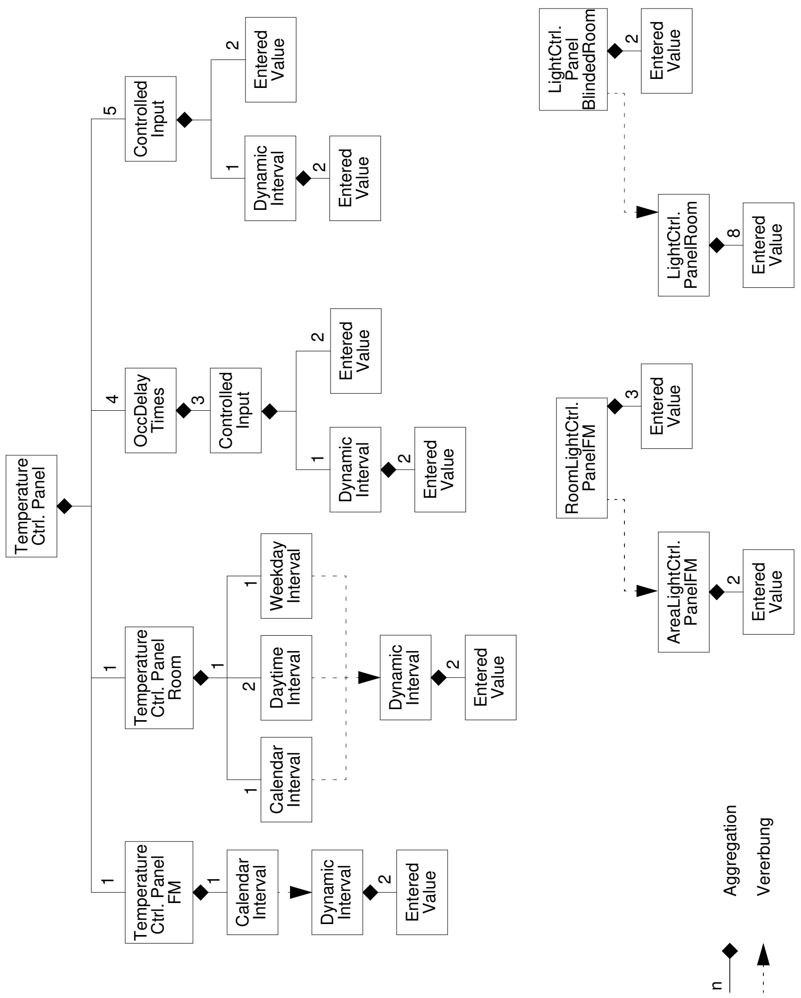

Abbildung 10: Klassenstruktur der Gruppe *HumanMachineInterface*

Global Signature

Domain $TEMP = \{-40 \ldots 100\} < INTEGER$

Intention	: This domain contains the possible temperature values in degrees Celsius that are considered in this problem specification.

Domain $LIGHT\_VALUES = \{1 \ldots 10000\} < NAT$

Intention	: This domain contains the possible light intensity values in lux that are considered in this problem specification.

Domain $BINARY = \{0, 1\} < NAT$

Intention	: This domain contains the values 0 and 1.

Domain $PERCENT = \{0 \ldots 100\} < NAT$

Intention	: This domain contains the natural numbers between 0 to 100.

Domain $THREE\_PT\_ACT\_RANGE = \{FORWARD, BACKWARD, STOP\}$

Intention	: This domain contains the possible control states of a three point actuator.

Domain $CALENDAR\_TIME$

Intention	: This domain contains the values that can be used for a calendar time, i.e. a date, it consists of a day and a month.

Domain $WEEKDAY = \{Mo, Tu, We, Th, Fr, Sa, Su\}$

Intention	: This domain contains the values that can be used for a week-day.

Domain $DAYTIME$

Intention	: This domain contains the possible day times in hour and minutes.

Timed Function $currentDate() \rightarrow CALENDAR\_TIME$

Intention	: This function represents the current date, i.e. the current day and month
Scope	: environment controlled, not visible for the machine

Timed Function $currentWeekday() \rightarrow WEEKDAY$

Intention	: This function represents the current day of the week.
Scope	: environment controlled, not visible for the machine

Timed Function $currentDaytime() \rightarrow DAYTIME$

Intention	: This function represents the current daytime in hour and minutes.
Scope	: environment controlled, not visible for the machine

Group Datatypes

Intention

This group contains all description classes that are used as datatypes. This means that the description classes in this group are only applied to the declaration of functions or predicates. They are not used for aggregation or inheritance.

Description Class LightScene

Intention

This class provides all definitions concerning a light scene for an arbitrary room. These are w.r.t. the DIC document:

- a name,
- a value of the light intensity of the ambient light of a room,
- an ordered list containing the installed lamps. The list defines the order in which the control system has to use the lamps to reach the given ambient light value. Since this list always consists of two entries we model this list by two functions firstAmbientIlluminaire and secondAmbientIlluminaire.

Aspects concerning a light scene of an office are presented in the derived description class LightSceneOffice.

Signature

Domain $LAMPS = \{WINDOW, WALL\}$

Intention	: This domain contains the installed lamps that can be used to establish a certain light scene. • $WINDOW$ stands for the window ceiling light and • $WALL$ for the ceiling light near the door to the hallway.

Timed Function $name() \rightarrow STRING$

Intention	: This function represents the name of the light scene.
Scope	: machine controlled, not visible for the environment

Timed Function $ambientIlluminance() \rightarrow LIGHT\_VALUES$

Intention	: This function represents the desired ambient light intensity of a room.
Scope	: machine controlled, not visible for the environment

Timed Function $firstAmbientLighting() \rightarrow LAMPS$

Intention	: This function represents the first lamp that should be used to reach the desired ambient light of a room.
Scope	: machine controlled, not visible for the environment

Timed Function $secondAmbientLighting() \rightarrow LAMPS$

Intention	: This function represents the second lamp that should be used to reach the desired ambient light of a room.
Scope	: machine controlled, not visible for the environment

Requirements/Specifications

Property P_LS_1

Formal	: $\Box(\neg(firstAmbientLighting = secondAmbientLighting))$
NL	: The first lamp that should be used to reach the desired ambient light of a room and the second lamp that should be used to reach the desired ambient light of a room are always different.

Description Class LightSceneOffice

Intention

This Class contains all the entities concerning a light scene for an office. In addition to the entities belonging to the basic class *LightScene* this class also contains the following entities.

- a value of the light intensity for the desk
- An ordered list containing the installed lamps. The list defines the order in which the control system has to use the lamps to reach the given light intensity value for the desk.

Note that it is not possible for the machine to control the task light. Therefore, it is not considered here!

Base Classes

Class *LightScene*

Intention	:	All definitions of description class *LightScene* are also provided here.

Signature

Timed Function $firstDeskLighting() \rightarrow LightScene :: LAMPS$

Intention	:	This function represents the first lamp that should be used to reach the desired desk light of a room.
Scope	:	machine controlled, not visible for the environment

Timed Function $secondDeskLighting() \rightarrow LightScene :: LAMPS$

Intention	:	This function represents the second lamp that should be used to reach the desired desk light of a room.
Scope	:	machine controlled, not visible for the environment

Requirements/Specifications

Property P_LSO_1

Formal	:	$\Box(\neg(firstDeskLighting = secondDeskLighting))$
NL	:	The first lamp that should be used to reach the desired desk light and the second lamp that should be used to reach the desired desk light are always different.

Group BasicHumanMachineInterface

Intention

This group contains all description classes that contain the basic definitions representing the interface between the machine and a person. This means here the possibility for a person to enter a value.

Description Class EnteredValue

Intention

This class provides all definitions to describe the entering of a value using an appropriate device (in our case this device is a control panel). For a value e that is entered (expressed by the function *entered*) a value c (expressed by the function *checked*) is determined using the function *nearestValue*. This function is intended to check the entered value e whether it is an allowed value and returns the value which is *nearest* to e.

It is assumed that the control panel already exists.

Formal Parameters

Domain *CHECKED_DOM*

Intention	: This domain contains the values that can be produced out of an entered value using the function *nearestValue*.

Signature

Timed Function $entered() \rightarrow STRING$

Intention	: This function represents the value entered by a person. The value is represented as a string.
Scope	: environment controlled, not visible for the machine

Timed Function $checked() \rightarrow CHECKED\_DOM$

Intention	: This function represents the value decided by the machine to use. *checked* is determined out of *entered* using the function *nearestValue*.
Scope	: environment controlled, visible for the machine

Timed Function $nearestValue(STRING) \rightarrow CHECKED\_DOM$

Intention	: This function returns the value which is the *nearest* one in the domain CHECKED_DOM w.r.t. the first argument.
Scope	: environment controlled, not visible for the machine

Timed Function $checkTime() \rightarrow TIME$

Intention	: This function represents the time needed to check a value, entered by a person, using the function *nearestValue*.
Scope	: environment controlled, not visible for the machine

Domain Knowledge

Property D_EV_1

Formal	: $\square(entered \triangleright^{nearestValue}_{checkTime} checked)$
NL	: The checked value is always derived from the entered value using the function *nearestValue*. A change of the entered value is always propagated within the time *checkTime*.

Description Class DynamicInterval

Intention

This class provides all definitions to describe a *dynamic interval*. An interval is a subset of a specific domain and is given by a start (object *start*) and an end point (object *end*). The domain is given by the formal parameter *INT_DOM*. Furthermore, a predicate *leq* is provided indicating whether a value of the considered domain is less or equal to another value.

Dynamic means that the values of the start and end point of the interval can change over time. Both values can be entered by a person. Therefore the start and end point are objects of the class *EnteredValue*.

Formal Parameters

Domain *INT_DOM*

Intention	: This domain contains the values the start and end point of the interval have to belong to.

Signature

Object $start : EnteredValue(CHECKED\_DOM = INT\_DOM)$

Intention	: This object represents the starting point of the interval that can be entered by a person. The entered value has to be an element of the domain *INT_DOM*.

Object $end : EnteredValue(CHECKED\_DOM = INT\_DOM)$

Intention	: This object represents the end point of the interval that can be entered by a person. The entered value has to be an element of the domain *INT_DOM*.

Predicate $leq : (INT\_DOM, INT\_DOM)$

Intention	: This predicate is true iff the first argument is *less* than or equal to the second argument.
Scope	: machine controlled, not visible for the environment

Description Class DaytimeInterval

Intention

This class provides all definitions to describe an interval of daytimes. The start and end point of the interval can change over time. Therefore, this class is derived from the description class *DynamicInterval*. Additionally, a timed predicate *actual* indicates whether the current daytime (*currentDaytime*) is in this interval or not.

Base Classes

Class $DynamicInterval(INT\_DOM = DAYTIME)$

Intention : All definitions of a dynamic interval are also provided here, using the domain containing the daytimes as interval domain.

Signature

Timed Predicate *actual*

Intention : This predicate is true iff the current daytime is in the daytime interval.

Scope : machine controlled, not visible for the environment

Domain Knowledge

Property P_DI_1

Formal :
$$\Box(actual \leftrightarrow (leq(start.checked, end.checked) \wedge leq(start.checked, currentDaytime) \wedge leq(currentDaytime, end.checked)) \vee (\neg leq(start.checked, end.checked) \wedge (leq(start.checked, currentDaytime) \vee leq(currentDaytime, end.checked))))$$

NL : The current daytime is in the daytime interval iff one of the following conditions is fulfilled:

- the start point of the intervall is less than or equal to the end point of the interval and the start point of the intervall is less than or equal to the current daytime and the current daytime is less than or equal to the end point of the interval;
- the start point of the intervall is not less than or equal to the end point of the interval and either is the start point of the intervall less than or equal to the current daytime or the current daytime is less than or equal to the end point of the interval.

Description Class WeekdayInterval

Intention

This class provides all definitions to describe an interval of week-days. The start and end point of the interval can change over time. Therefore, this class is derived from the description class *DynamicInterval*. Additionally, a timed predicate *actual* indicates whether the current week-day (*currentWeekday*) is in this interval or not.

Base Classes

Class $DynamicInterval(INT\_DOM = WEEKDAY)$

Intention : All definitions of a dynamic interval are also provided here, using the domain containing the week-days as interval domain.

Signature

Timed Predicate *actual*

Intention : This predicate is true iff the current week-day is in the week-day interval.

Scope : machine controlled, not visible for the environment

Domain Knowledge

Property P_WI_1

Formal :
$$\Box(actual \leftrightarrow (leq(start.checked, end.checked) \wedge leq(start.checked, currentWeekday) \wedge leq(currentWeekday, end.checked)) \vee (\neg leq(start.checked, end.checked) \wedge (leq(start.checked, currentWeekday) \vee leq(currentWeekday, end.checked))))$$

NL : The current week-day is in the week-day interval iff one of the following conditions is fulfilled:

- the start point of the intervall is less than or equal to the end point of the interval and the start point of the intervall is less than or equal to the current week-day and the current week-day is less than or equal to the end point of the interval;
- the start point of the intervall is not less than or equal to the end point of the interval and either is the start point of the intervall less than or equal to the current week-day or the current week-day is less than or equal to the end point of the interval.

Description Class CalendarInterval

Intention

This class provides all definitions to describe an interval of calendar times. The start and end point of the interval can change over time. Therefore, this class is derived from the description class *DynamicInterval*. Additionally, a timed predicate *actual* indicates whether the current date (*currentDate*) is in this interval or not.

Base Classes

Class $DynamicInterval(INT\_DOM = CALENDAR\_TIME)$

Intention : All definitions of a dynamic interval are also provided here, using the domain containing the calendar times as interval domain.

Signature

Timed Predicate *actual*

Intention : This predicate is true iff the current date is in the calendar interval.
Scope : machine controlled, not visible for the environment

Domain Knowledge

Property P_CI_1

Formal :
$$\Box(actual \leftrightarrow (leq(start.checked, end.checked) \land leq(start.checked, currentDate) \land leq(currentDate, end.checked)) \lor (\neg leq(start.checked, end.checked) \land (leq(start.checked, currentDate) \lor leq(currentDate, end.checked))))$$

NL : The current date is in the calendar interval iff one of the following conditions is fulfilled:

- the start point of the intervall is less than or equal to the end point of the interval and the start point of the intervall is less than or equal to the current date and the current date is less than or equal to the end point of the interval;
- the start point of the intervall is not less than or equal to the end point of the interval and either is the start point of the intervall less than or equal to the current date or the current date is less than or equal to the end point of the interval.

Description Class ControlledInput

Intention

This class contains all the definitions that are necessary to model the *controlled* input of a value by a user. This value is represented by the timed function *value*. *Controlled* means here that the facility manager prescribes an interval in which the value entered by a user has to be. Furthermore, the facility manager prescribes a default value (object *default*) which is valid as long as a user has not entered a value. As soon as a user has entered a value this becomes the actual value unless a user changes it by entering a new value. There is no other possibility to change the value.

Formal Parameters

Domain $INPUT\_DOM$

Intention : This domain contains all the values that can be entered for the boundaries of the interval the facility manager can prescribe, for the default value given by the facility manager, and for the value entered by a user.

Signature

Object $allowedRange : DynamicInterval(INT\_DOM = INPUT\_DOM)$

Intention : This object represents the range of possible user entries. This range is specified by the facility manager.

Object $default : EnteredValue(CHECKED\_DOM = INPUT\_DOM)$

Intention : This object represents the default value given by the facility manager. Note that by this declaration it is only guaranteed that the default value is an element of the domain $INPUT\_DOM$. It is not guaranteed that it is a value of the interval specified by the facility manager. This is expressed by the property D_CIN_3 of this class.

Object $userValue : EnteredValue(CHECKED\_DOM = INPUT\_DOM)$

Intention : This object represents the value entered by a user.
Note that by this declaration it is only guaranteed that the entered value is an element of the domain $INPUT\_DOM$. It is not guaranteed that it is a value of the interval specified by the facility manager. This is expressed by the property D_CIN_2 of this class.

Timed Function $value() \to INPUT\_DOM$

Intention : This function represents the currently valid value. It depends on the inputs of the facility manager and the input of a user.
Scope : environment controlled, visible for the machine

Timed Predicate *userValueEntered*

Intention : This predicate is true iff the user has entered a value.
Scope : environment controlled, not visible for the machine

Domain Knowledge

Initialization

Property I_CIN_1

Formal	: $\neg userValueEntered$
NL	: Initially a value has not been entered by a user.

Property I_CIN_2

Formal	: $value = default.checked$
NL	: The initially valid value is equal to the initial default value given by the facility manager.

End of Initialization

Property D_CIN_1

Formal	: $\Box(userValueEntered \rightarrow \Box userValueEntered)$
NL	: If a user has entered a value once, from this time point it will be valid that a value has been entered by a user.

Property D_CIN_2

Formal :
$$\begin{aligned}
&\Box(userValueEntered \rightarrow \\
&\quad (allowedRange.leq(userValue.checked, \\
&\qquad\qquad allowedRange.start.checked) \\
&\qquad \rightarrow value = allowedRange.start.checked) \wedge \\
&\quad (allowedRange.leq(allowedRange.end.checked, \\
&\qquad\qquad userValue.checked) \\
&\qquad \rightarrow value = allowedRange.end.checked) \wedge \\
&\quad ((\neg allowedRange.leq(userValue.checked, \\
&\qquad\qquad allowedRange.start.checked) \wedge \\
&\quad \neg allowedRange.leq(allowedRange.end.checked, \\
&\qquad\qquad userValue.checked)) \\
&\qquad \rightarrow value = userValue.checked))
\end{aligned}$$

NL : If a user has entered a value which is less than or equal to the lower bound of the allowed interval specified by the facility manager the currently valid value is equal to the lower bound of the interval specified by the facility manager.
If a user has entered a value which is greater than the upper bound of the allowed interval specified by the facility manager the currently valid value is equal to the upper bound of the interval specified by the facility manager.
Otherwise, if the entered value is within the interval specified by the facility manager, the currently valid value is equal to the value entered by a user.

Property D_CIN_3

Formal :
$$\begin{aligned}
&\Box(allowedRange.leq(allowedRange.start.checked, \\
&\qquad\qquad default.checked) \wedge \\
&\quad allowedRange.leq(default.checked, \\
&\qquad\qquad allowedRange.end.checked))
\end{aligned}$$

NL : The default value given by the facility manager has always to be within the allowed range specified by the facility manager.

Group Sensors

Intention

This group contains all description classes that contain definitions related to sensors.

Description Class NoMalfunctionSensor

Intention

This class provides all definitions to describe the behavior of a *sensor* if it has *no malfunction*.

It is distinguished between three values:

- the value of the phenomenon existing in the real world (which can never be visible for the machine) (timed function *envEntity*),
- the analogously measured value of this real world value (timed function *measuredEntity*), and
- the digitally measured value which is a conversion of the analog measured value (timed function *convertedEntity*).

Note that this class can be used to describe a sensor that already exists in the environment. This is expressed as follows: At first the value measured by the sensor, namely *convertedEntity*, has the scope *environment controlled, not visible for the machine*. Secondly, the properties D_NMS_1 and D_NMS_2 describing the behavior of the sensor are characterized as domain knowledge. Otherwise, if the sensor does not exist, they have to be requirements.

Formal Parameters

Domain *ENV_DOMAIN*

Intention	:	This domain contains the possible values of the measured phenomenon existing in the real world.

Domain *MEASURED_DOMAIN*

Intention	:	This domain contains the possible analogously measured values.

Domain *CONVERTED_DOMAIN*

Intention	:	This domain contains the possible digitally converted measured values.

Signature

Time Constant *reactionTime*

Intention	:	This time constant represents the *reaction time* of the sensor. The *reaction time* is the time between a change of the value in the real world (*envEntity*) and the time point the measured entity (*measuredEntity*) has reached 90 percent of the change.
Scope	:	environment controlled, not visible for the machine

Time Constant *conversionTime*

Intention : This time constant represents the *conversion time* of the sensor. The *conversion time* is the time needed to convert an analogously measured value (*measuredEntity*) into the corresponding digital value (*convertedEntity*).

Scope : environment controlled, not visible for the machine

Timed Function $envEntity() \rightarrow ENV\_DOMAIN$

Intention : This function represents the value of the measured phenomenon in the real world.
Note that this function represents the phenomenon as it is in the real world and not how it is seen by the machine.

Scope : environment controlled, not visible for the machine

Timed Function $measuredEntity() \rightarrow MEASURED\_DOMAIN$

Intention : This function represents the analogously measured value of the phenomeno existing in the real world.

Scope : environment controlled, not visible for the machine

Timed Function $convertedEntity() \rightarrow CONVERTED\_DOMAIN$

Intention : This function represents the digitally converted value of the measured value.

Scope : environment controlled, visible for the machine

Timed Function $modifyReaction(ENV\_DOMAIN) \rightarrow MEASURED\_DOMAIN$

Intention : This function represents the way in which a value of the phenomenon in the real world (*envEntity*) is modified during the reaction of the sensor resulting in an analogously measured value *measuredEntity*.

Scope : environment controlled, not visible for the machine

Timed Function $modifyConversion(MEASURED\_DOMAIN) \rightarrow CONVERTED\_DOMAIN$

Intention : This function represents the way in which an analogously measured value *measuredEntity* is modified during the conversion into a digital value (*convertedEntity*).

Scope : environment controlled, not visible for the machine

Domain Knowledge

Property D_NMS_1

Formal : $\Box(envEntity \rhd^{modifyReaction}_{reactionTime} measuredEntity)$

NL : The analogously measured value is always derived from the value of the phenomenon in the real world using the function *modifyReaction*. A change of the value in the real world is always propagated within the time *reactionTime*.

Property D_NMS_2

Formal : $\Box(measuredEntity \rhd^{modifyConversion}_{conversionTime} convertedEntity)$

NL : The digitally converted value is always derived from the analogously measured value using the function *modifyConversion*. A change of the value in the real world is always propagated within the time *conversionTime*.

Description Class MalfunctionSensor

Intention

This class provides all definitions to describe the behavior of a *sensor* if it has a *malfunction*.

In the case of a malfunction of a sensor, a default value is used that is determined by the facility manager.

Furthermore, the facility manager is informed about a malfunction.

Formal Parameters

Domain *MAL_CONVERTED_DOMAIN*

Intention : This domain contains the possible values the machine can use as sensor values in the case of a malfunction of the sensor.

Signature

Object *manualEnteredVal* : *EnteredValue*
$(CHECKED\_DOM = MAL\_CONVERTED\_DOMAIN)$

Intention : This object represents the value used in the case of a malfunction of this sensor. It is determined by the facility manager. The entered value has to be an element of the domain MAL_CONVERTED_DOMAIN.

Time Constant $T_{infDelay}$

Intention : This time constant represents the maximal delay between an important event concerning this sensor the facility manager has to be informed about (e.g. a malfunction or the repair of a malfunction) and the time point the facility manager is informed.
Scope : environment controlled, not visible for the machine

Time Constant T_{info}

Intention : This time constant represents the maximal time span the facility manager is informed about an important event concerning this sensor.
Scope : environment controlled, not visible for the machine

Timed Predicate *malDetEnv*

Intention : This predicate is true iff the sensor existing in the real world has a malfunction. Note that this predicate represents the value as it exists in the real world. This value is not visible for the machine.
Scope : environment controlled, not visible for the machine

Timed Predicate *malDetMach*

Intention : This predicate is true iff the machine assumes that the sensor existing in the real world has a malfunction. Note that this predicate represents the value as it seen by the machine.
Scope : machine controlled, not visible for the environment

Timed Predicate *FMInformed*

Intention : This predicate is true iff the facility manager is informed about an *important event* concerning this sensor.
An *important event* is either the detection of a malfunction of the sensor or that a malfunction of the sensor has been repaired.
The facility manager has to be informed by sending an email.
Scope : environment controlled, not visible for the machine

Requirements/Specifications

Initialization

Property I_MS_1

Formal : $\neg FMInformed$
NL : Initially the facility manager is not informed about an important event concerning a sensor.

End of Initialization

Property P_MS_1

Formal : $\Box([FMInformed] \rightarrow \Diamond_{\leq T_{info}} \neg FMInformed)$
NL : Whenever the facility manager becomes informed he is informed for at most T_{info} milliseconds.
By this requirement it is expressed, that the facility manager is not informed permanently, but that the information is a kind of signal, e.g. an email.

Property P_MS_2

Formal : $\Box([malDetMach] \rightarrow \Diamond_{\leq T_{infDelay}} FMInformed)$
NL : Whenever the machine assumes that a malfunction of the sensor has occurred, the facility manager is informed about this within $T_{infDelay}$ milliseconds.

Property P_MS_3

Formal : $\Box([\neg malDetMach] \rightarrow \Diamond_{\leq T_{infDelay}} FMInformed)$
NL : Whenever the machine assumes that the sensor is working correctly again after it has had a malfunction, the facility manager is informed about this within $T_{infDelay}$ milliseconds.

Property P_MS_4

Formal : $\Box(T_{infDelay} = 2000)$
NL : The maximal delay between the occurence of an event the facility manager has to be informed about and the time point he becomes informed is 2 seconds.

Property P_MS_5

Formal : $\Box(T_{info} = 10000)$
NL : The facility manager is informed for maximal 10 seconds.

Description Class Sensor

Intention

This class provides all definitions to describe the behavior of a *sensor*. It is composed of the behavior of a sensor if there is no malfunction and the case that there is a malfunction.

A malfunction of a sensor can be entered manually.

Formal Parameters

Domain ENV_DOMAIN

Intention : This domain contains the possible values of the measured phenomenon existing in the real world in the case that a sensor has no malfunction.

Domain $MEASURED\_DOMAIN$

Intention : This domain contains the possible analogously measured values in the case that a sensor has no malfunction.

Domain $CONVERTED\_DOMAIN$

Intention : This domain contains the possible digitally converted measured values in the case that a sensor has no malfunction.

Signature

Object $noMalSens : NoMalfunctionSensor(ENV\_DOMAIN = ENV\_DOMAIN, MEASURED\_DOMAIN = MEASURED\_DOMAIN, CONVERTED\_DOMAIN = CONVERTED\_DOMAIN)$

Intention : This object represents the part of the sensor describing the behavior of the sensor if it has no malfunction, i.e. if the machine assumes that it has no malfunction.

Object $malSens : MalfunctionSensor(MAL\_CONVERTED\_DOMAIN = CONVERTED\_DOMAIN)$

Intention : This object represents the part of the sensor describing the behavior of the sensor if it has a malfunction, i.e. if the machine assumes that it has a malfunction.
The possible digitally converted measured values ($CONVERTED\_DOMAIN$) are used as the possible values the machine can use as sensor values in the case of a malfunction of the sensor ($MAL\_CONVERTED\_DOMAIN$).

Object $manualEnteredMal : EnteredValue(CHECKED\_DOM = BINARY)$

Intention : This object represents a value entered manually by a person concerning a malfunction of the sensor. 1 stands for a malfunction, 0 stands for no malfunction.
Note that the value of this object is only meaningful if the timed predicate *stateManualEnteredMal* is true.

Timed Function $usedValue() \rightarrow CONVERTED\_DOMAIN$

Intention : This function models the current value used by the machine as the value of the sensor. This depends on whether there is a malfunction of the sensor or not.

Scope : machine controlled, not visible for the environment

Timed Predicate *stateManualEnteredMal*

Intention : This predicate is true iff it has been manually entered whether there is a malfunction of the sensor or whether there is no malfunction of it.
Note the difference between this predicate and the object *manualEnteredMal*.

Scope : machine controlled, not visible for the environment

Requirements/Specifications

Initialization

Property I_S_1

Formal : $\neg stateManualEnteredMal$

NL : Initially it has not been entered manually whether there is a malfunction of the sensor or whether there is no malfunction of it.

End of Initialization

Property P_S_1

Formal : $\Box(malSens.malDetMach \leftrightarrow (stateManualEnteredMal \wedge manualEnteredMal.checked = 1))$

NL : The machine assumes a malfunction of the sensor iff a malfunction has been entered manually.

Property P_S_2

Formal : $\Box(\neg malSens.malDetMach \rightarrow usedValue = noMalSens.convertedEntity)$

NL : Whenever the machine assumes that the sensor is working correctly the value used by the machine corresponds with the digitally converted version of the measured value.

Property P_S_3

Formal : $\Box(malSens.malDetMach \rightarrow usedValue = malSens.manualEnteredVal.checked)$

NL : Whenever the machine assumes that the sensor is not working correctly the value used by the machine corresponds with the value entered by the facility manager for this case.

Description Class RoomTempSensor

Intention

This class provides all definitions to describe the behavior of a *room temperature sensor*. This class is derived from the description class *Sensor*. The following specific parameters of a room temperature sensor are instantiated:

- the domains of the sensor
- the reaction and conversion time of the sensor

Both functions, *modifyReaction* and *modifyConversion*, have still to be provided by the domain expert.

Base Classes

Class $Sensor(ENV\_DOMAIN = REAL, MEASURED\_DOMAIN = REAL,$
$CONVERTED\_DOMAIN = ROOM\_TEMP)$

Intention : All definitions of description class *Sensor* are also provided here using the real numbers as the possible values of the phenomenon existing in the real world, using the real numbers as the possible analogously measured values, and using the values of the domain $ROOM\_TEMP$ as the possible digitally converted measured values.

Signature

Domain $ROOM\_TEMP = \{-40 \ldots 70\} < TEMP$

Intention : This domain contains the possible values measured by a room temperature sensor.

Domain Knowledge

Property D_RTS_1

Formal : $\Box(noMalSens.conversionTime = 160000)$
NL : The conversion time of a room temperature sensor is 160 seconds.

Property D_RTS_2

Formal : $\Box(noMalSens.reactionTime = 10000)$
NL : The reaction time of a room temperature sensor is 10 seconds.

Description Class OutdoorTempSensor

Intention

This class provides all definitions to describe the behavior of a *outdoor temperature sensor*. This class is derived from the description class *Sensor*. The following specific parameters of a outdoor temperature sensor are instantiated:

- the domains of the sensor
- the reaction and conversion time of the sensor

Both functions, *modifyReaction* and *modifyConversion*, have still to be provided by the domain expert.

Base Classes

Class $Sensor(ENV\_DOMAIN = REAL, MEASURED\_DOMAIN = REAL,$
$CONVERTED\_DOMAIN = OUT\_TEMP)$

Intention : All definitions of description class *Sensor* are also provided here using the real numbers as the possible values of the phenomenon existing in the real world, using the real numbers as the possible analogously measured values, and using the the values of the domain OUT_TEMP as the possible digitally converted measured values.

Signature

Domain $OUT\_TEMP = \{-40 \ldots 60\} < TEMP$

Intention : This domain contains the possible values measured by an outdoor temperature sensor.

Domain Knowledge

Property D_OTS_1

Formal : $\Box(noMalSens.conversionTime = 160000)$
NL : The conversion time of an outdoor temperature sensor is 160 seconds.

Property D_OTS_2

Formal : $\Box(noMalSens.reactionTime = 120000)$
NL : The reaction time of an outdoor temperature sensor is 120 seconds.

Description Class OutdoorLightSensor

Intention

This class provides all definitions to describe the behavior of a *outdoor light sensor*. This class is derived from the description class *Sensor*. The following specific parameters of a outdoor light sensor are instantiated:

- the domains of the sensor
- the reaction and conversion time of the sensor

Both functions, *modifyReaction* and *modifyConversion*, have still to be provided by the domain expert.

Base Classes

Class $Sensor(ENV\_DOMAIN = REAL, MEASURED\_DOMAIN = REAL, CONVERTED\_DOMAIN = OUT\_LIGHT\_VALUES)$

Intention : All definitions of description class *Sensor* are also provided here using the real numbers as the possible values of the phenomenon existing in the real world, using the real numbers as the possible analogously measured values, and using the values of the domain OUT_LIGHT_VALUES as the possible digitally converted measured values.

Signature

Domain $OUT\_LIGHT\_VALUES = \{1 \ldots 10000\} < LIGHT\_VALUES$

Intention : This domain contains the possible values measured by an outdoor light sensor.

Domain Knowledge

Property D_OLS_1

Formal : $\Box(noMalSens.conversionTime = 1000)$
NL : The conversion time of an outdoor light sensor is 1 second.

Property D_OLS_2

Formal : $\Box(noMalSens.reactionTime = 10)$
NL : The reaction time of a outdoor light sensor is 10 milliseconds.

Description Class WaterTempSensor

Intention

This class provides all definitions to describe the behavior of a *water temperature sensor*. This class is derived from the description class *Sensor*. The following specific parameters of a water temperature sensor are instantiated:

- the domains of the sensor
- the reaction and conversion time of the sensor

Both functions, *modifyReaction* and *modifyConversion*, have still to be provided by the domain expert.

Base Classes

Class $Sensor(ENV\_DOMAIN = REAL, MEASURED\_DOMAIN = REAL, CONVERTED\_DOMAIN = WATER\_TEMP)$

Intention : All definitions of description class *Sensor* are also provided here using the real numbers as the possible values of the phenomenon existing in the real world, using the real numbers as the possible analogously measured values, and using the values of the domain $WATER\_TEMP$ as the possible digitally converted measured values.

Signature

Domain $WATER\_TEMP = \{0 \ldots 100\} < TEMP$

Intention : This domain contains the possible values measured by a water temperature sensor.

Domain Knowledge

Property D_WTS_1

Formal : $\Box(noMalSens.conversionTime = 160000)$
NL : The conversion time of a water temperature sensor is 160 seconds.

Property D_WTS_2

Formal : $\Box(noMalSens.reactionTime = 10000)$
NL : The reaction time of a water temperature sensor is 10 seconds.

Description Class BinarySensor

Intention

This class provides all definitions to describe the behavior of a *binary sensor*. This class is derived from the description class *Sensor*. The following specific parameters of a binary sensor are instantiated:

- the domains of the sensor: a binary sensor is a sensor that returns a 0 or a 1,
- the conversion time of the sensor,
- the transformation functions *modifyReaction* and *modifyConversion* are defined explicitely.

Base Classes

Class $Sensor(ENV\_DOMAIN = BINARY,$
$MEASURED\_DOMAIN = BINARY,$
$CONVERTED\_DOMAIN = BINARY)$

Intention : All definitions of description class *Sensor* are also provided here using the values 0 and 1 as the possible values of the phenomenon existing in the real world, using the values 0 and 1 as the possible analogously measured values, and using the values 0 and 1 as the possible digitally converted measured values.

Domain Knowledge

Property D_BS_1

Formal : $\square(noMalSens.conversionTime = 0)$
NL : The conversion time of a binary sensor is 0 milliseconds.

Property D_BS_2

Formal : $\square(\forall b \in BINARY : noMalSens.modifyReaction(b) = b)$
NL : For a binary sensor the function representing the way in which a value in the real world (*envEntity*) is modified during the reaction of the sensor, resulting in an analogously measured value (*measuredEntity*), is the identity.

Property D_BS_3

Formal : $\square(\forall b \in BINARY : noMalSens.modifyConversion(b) = b)$
NL : For a binary sensor the function representing the way in which an analogously measured value (*measuredEntity*) is modified during the conversion into a digital value (*convertedEntity*) is the identity.

Description Class LightSensor

Intention

This class provides all definitions of a sensor determining whether a light is on or not. Note that this is not a sensor measuring the illuminance. Therefore, *on* means that the circuit of the light is closed.

Base Classes

Class *BinarySensor*

Intention : All definitions of a binary sensor are also provided here.

Domain Knowledge

Property D_L_1

Formal : $\square(noMalSens.reactionTime = 10)$
NL : The reaction time of a light sensor is 10 ms.

Description Class MotionDetector

Intention

This class provides all definitions to describe the behavior of a motion detector. A motion detector is derived from a binary sensor. Here additionally the reaction time of a motion detector is specified.

A motion detector returns a 1 if a person is moving in the range of the detector.

Base Classes

Class *BinarySensor*

Intention	:	All definitions of description class *BinarySensor* are also provided here.

Domain Knowledge

Property D_MD_1

Formal	:	$\Box(noMalSens.reactionTime = 1000)$
NL	:	The reaction time of a motion detector is 1 s.

Group Actuators

Intention

This group contains all description classes that contain definitions related to actuators.

Description Class NoMalfunctionActuator

Intention

This class provides all definitions to describe an *actuator* if it has *no malfunction*.

It is distinguished between two values:

- the value of the phenomenon as it exists in the real world (*envEntity*) (which can never be visible for the machine or controlled directly by the machine), and
- the value adjusted by the machine (*adjustedEntity*).

Note that this class can be used to describe an actuator that already exists in the environment. Therefore the value adjusted by the actuator, namely *adjustedEntity*, has the scope *machine controlled, visible for the machine*. If the actuator does not exist, i.e. the actuator itself has to be developed, the scope of *adjustedEntity* has to be *environment controlled, not visible for the machine*.

Formal Parameters

Domain *ENV_DOMAIN*

Intention	: This domain contains the possible values of the phenomenon existing in the real world.

Domain *ADJUSTED_DOMAIN*

Intention	: This domain contains the possible values that can be adjusted by the machine.

Signature

Time Constant *reactionTime*

Intention	: This time constant represents the *reaction time* of the actuator. The *reaction time* is the time between a change of the value adjusted by the machine (*adjustedEntity*) and the time point the value of the actuator in the real world (*envEntity*) changes its value *correspondingly*.
Scope	: environment controlled, not visible for the machine

Timed Function $envEntity() \rightarrow ENV\_DOMAIN$

Intention	: This function represents the value of the phenomenon, i.e. the actuator, existing in the real world. Note that this function represents the situation as it is in the real world and not how it is adjusted and seen by the machine.
Scope	: environment controlled, not visible for the machine

Timed Function $adjustedEntity() \rightarrow ADJUSTED\_DOMAIN$

Intention	: This function represents the value that is adjusted by the machine.
Scope	: machine controlled, visible for the environment

Description Class MalfunctionActuator

Intention

This class provides all definitions to describe the behavior of an *actuator* if it has a *malfunction*.

The facility manager is informed about a malfunction.

Signature

Time Constant $T_{infDelay}$

Intention	: This time constant represents the maximal delay between an important event concerning this actuator the facility manager has to be informed about (e.g. a malfunction or the repair of a malfunction) and the time point the facility manager is informed.
Scope	: environment controlled, not visible for the machine

Time Constant T_{info}

Intention	: This time constant represents the maximal time span the facility manager is informed about an important event concerning this actuator.
Scope	: environment controlled, not visible for the machine

Timed Predicate *malDetEnv*

Intention	: This predicate is true iff the actuator existing in the real world has a malfunction. Note that this predicate represents the value as it exists in the real world. This value is not visible for the machine.
Scope	: environment controlled, not visible for the machine

Timed Predicate *malDetMach*

Intention	: This predicate is true iff the machine assumes that the actuator existing in the real world has a malfunction. Note that this predicate represents the value as it seen by the machine.
Scope	: machine controlled, not visible for the environment

Timed Predicate *FMInformed*

Intention	: This predicate is true iff the facility manager is informed about an *important event* concerning this actuator. An *important event* is either the detection of a malfunction of the actuator or that a malfunction of the actuator has been repaired. The facility manager has to be informed by sending an email.
Scope	: environment controlled, not visible for the machine

Requirements/Specifications

Initialization

Property I_MA_1

Formal : $\neg FMInformed$

NL : Initially the facility manager is not informed about an important event concerning this actuator.

End of Initialization

Property P_MA_1

Formal : $\Box([FMInformed] \rightarrow \Diamond_{\leq T_{info}} \neg FMInformed)$

NL : Whenever the facility manager becomes informed he is informed for at most T_{info} milliseconds.
By this requirement it is expressed, that the facility manager is not informed permanently, but that the information is a kind of signal, e.g. an email.

Property P_MA_2

Formal : $\Box([malDetMach] \rightarrow \Diamond_{\leq T_{infDelay}} FMInformed)$

NL : Whenever the machine assumes that a malfunction of the actuator has occurred, the facility manager is informed about this within $T_{infDelay}$ milliseconds.

Property P_MA_3

Formal : $\Box([\neg malDetMach] \rightarrow \Diamond_{\leq T_{infDelay}} FMInformed)$

NL : Whenever the machine assumes that the actuator is working correctly again after it has had a malfunction, the facility manager is informed about this within $T_{infDelay}$ milliseconds.

Property P_MA_4

Formal : $\Box(T_{infDelay} = 2000)$

NL : The maximal delay between the occurence of an event the facility manager has to be informed about and the time point he becomes informed is 2 seconds.

Property P_MA_5

Formal : $\Box(T_{info} = 10000)$

NL : The facility manager is informed for maximal 10 seconds.

Description Class Actuator

Intention

This class provides all definitions to describe an *actuator*. It is composed of the behavior of an actuator if there is no malfunction and the case that there is a malfunction.

Formal Parameters

Domain *ENV_DOMAIN*

Intention : This domain contains the possible values of the phenomenon existing in the real world in the case that a sensor has no malfunction.

Domain *ADJUSTED_DOMAIN*

Intention : This domain contains the possible values that can be adjusted by the machine in the case that a sensor has no malfunction.

Signature

Object *noMalAct* : *NoMalfunctionActuator*(
ENV_DOMAIN = *ENV_DOMAIN*,
ADJUSTED_DOMAIN = *ADJUSTED_DOMAIN*)

Intention : This object represents the part of the actuator describing the behavior of the actuator if it has no malfunction, i.e. if the machine assumes that it has no malfunction.

Object *malAct* : *MalfunctionActuator*

Intention : This object represents the part of the actuator describing the behavior of the actuator if it has a malfunction, i.e. if the machine assumes that it has a malfunction.

Object *manualEnteredMal* : *EnteredValue*(*CHECKED_DOM* = *BINARY*)

Intention : This object represents a value entered manually by a user concerning a malfunction of the actuator. 1 stands for a malfunction, 0 stands for no malfunction.
Note that the value of this object is only meaningful if the timed predicate stateManualEnteredMal is true.

Timed Predicate *stateManualEnteredMal*

Intention : This predicate is true iff it has been manually entered whether there is a malfunction of the actuator or whether there is no malfunction of it.
Note the difference between this predicate and the object *manualEnteredMal*.

Scope : machine controlled, not visible for the environment

Requirements/Specifications

Initialization

Property I_A_1

Formal	: $\neg stateManualEnteredMal$
NL	: Initially it has not been entered manually whether there is a malfunction of the actuator or whether there is no malfunction of it.

End of Initialization

Property P_A_1

Formal	: $\Box(malAct.malDetMach \leftrightarrow (stateManualEnteredMal \wedge manualEnteredMal.checked = 1))$
NL	: The machine assumes a malfunction of the actuator iff a malfunction has been entered manually.

Description Class ToggleActuator

Intention

This class provides all definitions to describe a toggle actuator. A toggle actuator is an actuator that can send a signal. This is modeled here by setting the adjusted value to 1. If there is no signal the adjusted value is 0.

Note that this class can be used to describe an actuator that already exists in the environment. Therefore, the properties D_TA_1 and D_TA_2 are characterized as domain knowledge. Otherwise, if the actuator does not exist, they have to be requirements/specifications.

Base Classes

Class $Actuator(ENV\_DOMAIN = BINARY,$
$ADJUSTED\_DOMAIN = BINARY)$

Intention	: All definitions of description class *Actuator* are also provided here using the values 0 and 1 as the possible values of the phenomenon existing in the real world and using the values 0 and 1 as the possible values that can be adjusted by the machine.

Domain Knowledge

Property D_TA_1

Formal	: $\Box((noMalAct.envEntity = 1) \wedge [noMalAct.adjustedEntity = 1] \rightarrow \Diamond_{\leq noMalAct.reactionTime}(noMalAct.envEntity = 0))$
NL	: If the value of the entity in the real world (*envEntity*) is 1 and a toggle signal arrives, i.e. the adjusted value (*adjustedEntity*) changes from 0 to 1, then within *reactionTime* milliseconds the value of the entity in the real world (*envEntity*) is 0.

Property D_TA_2

Formal	: $\Box((noMalAct.envEntity = 0 \wedge [noMalAct.adjustedEntity = 1]) \rightarrow \Diamond_{\leq noMalAct.reactionTime}(noMalAct.envEntity = 1))$
NL	: If the value of the entity in the real world (*envEntity*) is 0 and a toggle signal arrives, i.e. the adjusted value (*adjustedEntity*) changes from 0 to 1, then within *reactionTime* milliseconds the value of the entity in the real world (*envEntity*) is 1.

Description Class Pulse

Intention

This class provides all definitions to describe the *pulse line* of a light by which it can be switched on and off. This class is derived from the description class *ToggleActuator*. Here, the reaction time of a pulse is specified.

Base Classes

Class *ToggleActuator*

Intention	: All definitions of description class *ToggleActuator* are also provided here.

Domain Knowledge

Property D_PUL_1

Formal	: $\Box(noMalAct.reactionTime = 10)$
NL	: The reaction time of a pulse line of a light is 10 milliseconds.

Description Class RangeActuator

Intention

This class provides all definitions to describe a *range actuator*. A range actuator is an actuator where a range of values, i.e. a set containing at least two values, can be adjusted. Consider in contrast to this the description class ToggleActuator.

Note that this class can be used to describe an actuator that already exists in the environment. Therefore, the property D_RA_1 is characterized as domain knowledge. Otherwise, if the actuator does not exist, it has to be a requirement/specification.

Formal Parameters

Domain *ENV_DOMAIN*

Intention	: This domain contains the possible values of the phenomenon existing in the real world in the case that a sensor has no malfunction.

Domain *ADJUSTED_DOMAIN*

Intention	: This domain contains the possible values that can be adjusted by the machine in the case that a sensor has no malfunction.

Base Classes

Class $Actuator(ENV\_DOMAIN = ENV\_DOMAIN, ADJUSTED\_DOMAIN = ADJUSTED\_DOMAIN)$

Intention	: All definitions of description class *Actuator* are also provided here.

Signature

Timed Function $modifyReaction(ADJUSTED\_DOMAIN) \rightarrow ENV\_DOMAIN$

Intention	: This function represents the way in which a value adjusted by the machine (*adjustedEntity*) is modified during the reaction of the actuator resulting in the value of the entity in the real world (*envEntity*).
Scope	: environment controlled, not visible for the machine

Domain Knowledge

Property D_RA_1

Formal	: $\Box(noMalAct.adjustedEntity \rhd^{modifyReaction}_{noMalAct.reactionTime} noMalAct.envEntity)$
NL	: The value of the phenomenon in the real world is always derived from the value adjusted by the machine using the function *modifyReaction*. A change of the value adjusted by the machine is always propagated within the time *reactionTime*.

Description Class Dimmer

Intention

This class provides all definitions to describe the *dimmer* of a dimmable light. This class is derived from the description class *RangeActuator*. The following specific parameters of a dimmer are instantiated:

- the domains of the sensor
- the reaction time

The function *modifyReaction* has still to be provided by the domain expert.

Base Classes

Class $RangeActuator(ENV\_DOMAIN = PERCENT,$
$ADJUSTED\_DOMAIN = PERCENT)$

Intention : All definitions of description class *RangeActuator* are also provided here using the values 0 to 100 (percent) as the possible values of the phenomenon existing in the real world and using the values 0 to 100 (percent) as the possible values that can be adjusted by the machine.

Domain Knowledge

Property D_DIM_1

Formal : $\Box(noMalAct.reactionTime = 10)$
NL : The reaction time of a *dimmer* is 10 milliseconds.

Description Class BinaryActuator

Intention

This class provides all definitions to describe a *binary actuator*. This class is derived from the description class *RangeActuator*. The following specific parameters of a binary actuator are instantiated:

- the domains of the actuator: a binary actuator is a range actuator that can adjust the values 0 and 1,
- the transformation function *modifyReaction*

Note that this class can be used to describe an actuator that already exists in the environment. Therefore, the property D_BA_1 is characterized as domain knowledge. Otherwise, if the actuator does not exist, it has to be a requirement/specification.

Base Classes

Class $RangeActuator(ENV\_DOMAIN = BINARY,$
$ADJUSTED\_DOMAIN = BINARY)$

Intention : All definitions of description class *RangeActuator* are also provided here using the values 0 and 1 as the possible values of the entity existing in the real world and using the values 0 and 1 as the possible values that can be adjusted by the machine.

Domain Knowledge

Property D_BA_1

Formal : $\Box(\forall b \in BINARY : modifyReaction(b) = b)$
NL : For a binary actuator the function representing the way in which a value adjusted by the machine (*adjustedEntity*) is modified during the reaction of the actuator, resulting in a value of the actuator in the real world (*envEntity*), is the identity.

Description Class ControlSystemActive

Intention

This class provides all definitions to describe the signal *control system is active* that can be sent by the machine in order to indicate that the machine is still active. In addition to the definitions of the description class *BinaryActuator*, here the reaction time of this signal is specified.

Base Classes

Class *BinaryActuator*

Intention	: All definitions of description class *BinaryActuator* are also provided here.

Domain Knowledge

Property D_CSA_1

Formal	: $\square(noMalAct.reactionTime = 10)$
NL	: The reaction time of a signal *control system is active* is 10 ms.

Description Class ThreePointActuator

Intention

This class provides all definitions to describe a *three point actuator*. This class is derived from the description class *RangeActuator*. The following specific parameters of a three point actuator are instantiated:

- the domains of the actuator: a three point actuator is an actuator which can adjust three control states: *forward*, *backward*, and *stop*,
- the transformation function *modifyReaction*

Note that this class can be used to describe an actuator that already exists in the environment. Therefore, the property D_TPA_1 is characterized as domain knowledge. Otherwise, if the actuator does not exist, it has to be a requirement/specification.

Base Classes

Class $RangeActuator(ENV\_DOMAIN = THREE\_PT\_ACT\_RANGE,$
$ADJUSTED\_DOMAIN = THREE\_PT\_ACT\_RANGE)$

Intention	: All definitions of description class *RangeActuator* are also provided here using the values *forward*, *backward*, and *stop* as the possible values of the actuator existing in the real world and using the values *forward*, *backward*, and *stop* as the possible values that can be adjusted by the machine.

Domain Knowledge

Property D_TPA_1

Formal	: $\square(\forall s \in THREE\_PT\_ACT\_RANGE : modifyReaction(s) = s)$
NL	: For a three point actuator the function representing the way in which a value adjusted by the machine (*adjustedEntity*) is modified during the reaction of the actuator, resulting in a value of the actuator in the real world (*envEntity*), is the identity.

Description Class RadiatorValve

Intention

This class provides all definitions to describe a radiator valve. In addition to the definitions of the description class *ToggleActuator*, here the reaction time of a radiator valve is specified.

Base Classes

Class *ThreePointActuator*

Intention	: All definitions of description class *ThreePointActuator* are also provided here.

Domain Knowledge

Property D_RV_1

Formal	: $\Box(noMalAct.reactionTime = 60000)$
NL	: The reaction time of a radiator valve is 60 seconds.

Description Class PushButton

Intention

This class provides all definitions concerning the control of a light using pushbuttons. The number of pushbuttons is not relevant for the behavior. Note further that the push buttons are not visible for the machine.

Signature

Time Constant T_PB

Intention	: This time constant represents the delay between a button push and the change of the light.
Scope	: environment controlled, not visible for the machine

Timed Predicate *light*

Intention	: This predicate represents the status of the light existing in the real world. The predicate is true iff the light is on. Note that *on* here means that the circuit of the light is closed.
Scope	: environment controlled, not visible for the machine

Timed Predicate *pushBut*

Intention	: This predicate represents the status of the push buttons existing in the real world. The predicate is true iff one push button is pushed.
Scope	: environment controlled, not visible for the machine

Domain Knowledge

Property D_PB_1

Formal	: $\Box(\neg light \wedge [pushBut] \rightarrow \Diamond_{\leq T\_PB} light)$
NL	: Whenever the light is off and any push button is pressed, the light changes to on within time T_PB.

Property D_PB_2

Formal	: $\Box(light \wedge [pushBut] \rightarrow \Diamond_{\leq T\_PB} \neg light)$
NL	: Whenever the light is on and any push button is pressed, the light changes to off within time T_PB.

Group CombinedSensorsAndActuators

Intention

This group contains all description classes in which several sensors or actuators are combined.

Description Class SunBlind

Intention

This class provides all definitions to describe a *sun blind*. A sun blind combines a *sun blind lifter* and a *blind rotator*. Both, lifter and rotator, are controlled using the same three point actuator. Additionally, a sun blind contains two sensors recognizing whether the blind has reached the upper or lower end.

To model the lifting of the blind and the rotating of the blind blades several states are introduced: *moveUp*, *moveDown*, *moveStop*, *rotateStatus*, *rotateIncrease*, *rotateDecrease*, and *rotateStop*.

Signature

Domain $DEGREE = \{0 \ldots 180\} < NAT$

Intention	: This domain contains the possible values of the blind rotator, namely 0 to 180 degrees.

Object $blind : ThreePointActuator$

Intention	: This object represents the actuator by which the blind can be lifted and rotated.

Object $upperEndContact : BinarySensor$

Intention	: This object represents the end contact detecting whether the blind is lifted completely up or not.

Object $lowerEndContact : BinarySensor$

Intention	: This object represents the end contact detecting whether the blind is lifted completely down or not.

Timed Function $rotateStatus() \rightarrow DEGREE$

Intention	: This function represents the current state of the blind rotator, i.e. the current position of the blind blades.
Scope	: environment controlled, not visible for the machine

Timed Predicate *moveUp*

Intention	: This predicate is true iff the blind is moving up.
Scope	: environment controlled, not visible for the machine

Timed Predicate *moveDown*

Intention	: This predicate is true iff the blind is moving down.
Scope	: environment controlled, not visible for the machine

Timed Predicate *moveStop*

Intention	: This predicate is true iff the blind stops, i.e. it is neither moving down nor up.
Scope	: environment controlled, not visible for the machine

Timed Predicate *rotateIncrease*

Intention	: This predicate is true iff the blind blades are rotated in the direction of 0 degree.
Scope	: environment controlled, not visible for the machine

Timed Predicate *rotateDecrease*

Intention	: This predicate is true iff the blind blades are rotated in the direction of 180 degree.
Scope	: environment controlled, not visible for the machine

Timed Predicate *rotateStop*

Intention	: This predicate is true iff the blind blades do not rotate in any direction. Note that this does not implies that the current rotating angle of the blades is 0 or 180 degrees.
Scope	: environment controlled, not visible for the machine

Domain Knowledge

Property D_SBL_1

Formal : $\Box((moveUp \leftrightarrow (\neg moveDown \wedge \neg moveStop)) \wedge (moveDown \leftrightarrow (\neg moveUp \wedge \neg moveStop)) \wedge (moveStop \leftrightarrow (\neg moveUp \wedge \neg moveDown)))$

NL : The blind lifter is always in exactly one of the three possible states *moveUp*, *moveDown*, *moveStop*.

Property D_SBL_2

Formal : $\Box((rotateIncrease \leftrightarrow (\neg rotateDecrease \wedge \neg rotateStop)) \wedge (rotateDecrease \leftrightarrow (\neg rotateIncrease \wedge \neg rotateStop)) \wedge (rotateStop \leftrightarrow (\neg rotateIncrease \wedge \neg rotateDecrease)))$

NL : The blind rotator is always in exactly one of the three possible states *rotateIncrease*, *rotateDecrease*, *rotateStop*.

Property D_SBL_3

Formal : $\Box(moveDown \rightarrow rotateStatus = 0)$

NL : If the blinds are moved down the rotating state, i.e. the angle of the blades is 0 degree.

Property D_SBL_4

Formal : $\Box(moveUp \rightarrow rotateStatus = 180)$

NL : If the blinds are moved up the rotating state, i.e. the angle of the blades is 180 degrees.

Property D_SBL_5

Formal : $\Box(blind.noMalAct.envEntity = STOP \rightarrow moveStop \wedge rotateStop)$

NL : Whenever the status of the sun blind is set to stop, the blind is neither lifted in any direction nor are the blades rotated in any direction.

Property D_SBL_6

Formal : $\Box(((blind.noMalAct.envEntity = FORWARD) \wedge (rotateStatus = 180) \wedge upperEndContact.noMalSens.envEntity = 0) \rightarrow (moveUp \wedge rotateStop))$

NL : Whenever the status of the sun blind is set to FORWARD and the rotating state of the blades, i.e. the angle, is 180 degrees and the blind is not at the upper end, the blind is lifted up and the blades are not rotating.

Property D_SBL_7

Formal : $\Box(((blind.noMalAct.envEntity = BACKWARD) \wedge (rotateStatus = 0) \wedge lowerEndContact.noMalSens.envEntity = 0) \rightarrow (moveDown \wedge rotateStop))$

NL : Whenever the status of the sun blind is set to BACKWARD and the rotating state of the blades, i.e. the angle, is 0 degrees and the blind is not at the lower end, the blind is lifted down and the blades are not rotating.

Property D_SBL_8

Formal : $\Box(((blind.noMalAct.envEntity = FORWARD) \wedge \neg(rotateStatus = 180)) \rightarrow (moveStop \wedge rotateIncrease))$

NL : Whenever the status of the sun blind is set to FORWARD and the rotating state of the blades, i.e. the angle, is not 180 degrees the blind is not lifted up or down in any direction but rotated in the direction of 180 degrees.

Property D_SBL_9

Formal : $\Box(((blind.noMalAct.envEntity = BACKWARD) \wedge \neg(rotateStatus = 0)) \rightarrow (moveStop \wedge rotateDecrease))$

NL : Whenever the status of the sun blind is set to BACKWARD and the rotating state of the blades, i.e. the angle, is not 0 degree the blind is not lifted up or down in any direction but rotated in the direction of 0 degrees.

Property D_SBL_{10}

NonFormal : If the rotating angle of the blind blades is x degrees light can enter from an angle of x degrees from vertical.

Description Class Radiator

Intention

This class provides all the definitions to describe a radiator. A radiator consists of a radiator valve and of three water temperature sensors measuring the input, mean, and output water temperature, respectively.

Signature

Object *waterInTemp* : *WaterTempSensor*

Intention : This object represents a sensor to measure the input water temperature of a radiator.

Object *waterOutTemp* : *WaterTempSensor*

Intention : This object represents a sensor to measure the output water temperature of a radiator.

Object *meanTemp* : *WaterTempSensor*

Intention : This object represents a sensor to measure the mean radiator temperature of a radiator.

Object *valveAct* : *RadiatorValve*

Intention : This object represents the radiator control valve of a radiator.

Description Class ControlledLight

Intention

This description class provides all definitions of sensors and actuators relevant for the control of a light that is not dimmable. This installation includes:

- a light sensor indicating whether the light is on or off (object *ls*)
- an actuator allowing the machine to switch on and off the light (object *pulse*)
- an actuator allowing the machine to signal that it is still active (object *csa*)
- a push button allowing a person to switch on and off the light (object *pb*) Note that a push of a button is not directly visible for the machine but only by the light sensor.

Furthermore, it is defined under which conditions a light is controlled *manually*.

Signature

Object *ls* : *LightSensor*

Intention : This object represents the light sensor indicating whether the light is on or off.

Object *pulse* : *Pulse*

Intention : This object represents the actuator the machine uses to switch the light on or off.

Object *csa* : *ControlSystemActive*

Intention : This object represents the actuator the machine uses to signal that it is still active.

Object *pb* : *PushButton*

Intention : This object represents the actuatori.e. the push button, a person can use to switch the light on or off. Note that a push of a button is not directly visible for the machine.

Timed Predicate *manualControl*

Intention : This predicate is true iff the machine assumes that the light is controlled manually.

Scope : machine controlled, not visible for the environment

Domain Knowledge

Property D_CL_1

Formal : $\Box(ls.noMalSens.envEntity = pulse.noMalAct.envEntity)$

NL : The phenomenon in the real world which is measured by the light sensor is identical with the phenomenon adjusted by the pulse actuator.

Property D_CL_2

Formal : $\Box((ls.noMalSens.envEntity = 1) \leftrightarrow pb.light)$
NL : The phenomenon in the real world which is measured by the light sensor is identical with the phenomenon adjusted by the push buttons.

Requirements/Specifications

Initialization

Property I_CL_1

Formal : $\neg manualControl$
NL : Initially the light is not controlled manually.

End of Initialization

Property P_CL_1

Formal : $\Box((\neg\blacklozenge_{\leq pulse.noMalAct.reactionTime}[pulse.noMalAct.adjustedEntity = 1] \wedge ([ls.noMalSens.convertedEntity = 1] \vee [ls.noMalSens.convertedEntity = 0])) \rightarrow manualControl)$
NL : If no pulse has been sent by the machine for the last *pulse.noMalAct.reactionTime* milliseconds and the measured value of the light sensor changes the light is controlled manually.

Property P_CL_2

Formal : $\Box(\blacklozenge_{\leq pulse.noMalAct.reactionTime}[pulse.noMalAct.adjustedEntity = 1] \rightarrow \neg manualControl)$
NL : If a pulse has been sent by the machine within the last *pulse.noMalAct.reactionTime* milliseconds the light is not controlled manually.

Property P_CL_3

Formal : $\Box([csa.noMalAct.adjustedEntity = 1] \rightarrow \Diamond_{\leq 1000} [csa.noMalAct.adjustedEntity = 0])$
NL : Whenever the signal *control system is active* is set to 1 it is set within one second to 0.

Property P_CL_4

Formal : $\Box([csa.noMalAct.adjustedEntity = 0] \rightarrow \Diamond_{\leq 59000} [csa.noMalAct.adjustedEntity = 1])$
NL : Whenever the signal *control system is active* is set to 0 it is set within 59 seconds to 1.

Description Class DimmableControlledLight

Intention

This description class provides all definitions of sensors and actuators relevant for the control of a dimmable light. This installation includes in addition to a controlled light:

- a dimmer, i.e. an actuator for the control of the illuminance of the light if it is on (object *dimmer*).
  Note that this dimmer can not switch on or off the light, i.e. even if the dimmer value is zero the light can have the status on, i.e. the circuit of the light is closed. Thus, the dimmer sets the illuminance in the case that the light is on.

Base Classes

Class *ControlledLight*

Intention : All definitions of description class BinarySensor are also provided here.

Signature

Object *dimmer* : *Dimmer*

Intention : This object represents the light actuator, i.e. the dimmer, the machine uses to set the illuminance of a light.

Group HumanMachineInterface

Intention

This group contains all description classes that contain definitions representing the interface between the machine and a person. These description classes are on a more abstract level than the description classes of group *BasicHumanMachineInterface*.

Description Class OccDelayTimes

Intention

This class provides all definitions to describe the values of the three time delays *entryTime*, dt_2, and *exitTime*, used to determine occupancy for the temperature control. The values of the three time delays depend on the inputs given by a user and the facility manager. These time delays can be set for the different temperature operation modes *NON_HEATING*, *OFF*, *STANDBY*, *COMFORT* (see description class *TemperatureCtrlPanel*).

This description class provides a general frame to deal with the three time delays in a common setting.

Signature

Object *entryTime* : *ControlledInput*(*INPUT_DOM* = *TIME*)

Intention : This object represents the current value of the time delay dt_1 (see description class *Occupancy*) depending on the values entered by the facility manager and the room user.

Object dt_2 : *ControlledInput*(*INPUT_DOM* = *TIME*)

Intention : This object represents the current value of the time delay dt_2 (see description class *Occupancy*) depending on the values entered by the facility manager and the room user.

Object *exitTime* : *ControlledInput*(*INPUT_DOM* = *TIME*)

Intention : This object represents the current value of the time delay dt_3 (see description class *Occupancy*) depending on the values entered by the facility manager and the room user.

Requirements/Specifications

Property P_ODT_1

Formal : $entryTime.default.checked = 60000$
NL : The initial default value of time *entryTime* given by the facility manager is 1 minute.

Property P_ODT_2

Formal : $dt_2.default.checked = 600000$
NL : The initial default value of time dt_2 given by the facility manager is 10 minutes.

Property P_ODT_3

Formal : $exitTime.default.checked = 1800000$
NL : The initial default value of time time *exitTime* given by the facility manager is 30 minutes.

Description Class AreaLightCtrlPanelFM

Intention

This class provides all the definitions of entities concerning the control of the light in an area for which the facility manager can enter a value using the control panel. An area is here the generalization of rooms and hallways. This is

- a possibility to switch on or off all lights in a certain area.
- a possibility to determine the time a certain area has to be unoccupied continuously before the lights are switched off.

Signature

Object $lightOn : EnteredValue(CHECKED\_DOM = BINARY)$

Intention : This object represents whether the facility manager has turned all lights on or off in a certain area. $lightOn.checked = 1$ means that the lights have to be on; $lightOn.checked = 0$ means that the lights have to be off.

Object $unoccDelay : EnteredValue(CHECKED\_DOM = TIME)$

Intention : This object represents the time an area has to be unoccupied continuously before the lights in this area are switched off.

Requirements/Specifications

Initialization

Property I_LCPFM_1

Formal : $unoccDelay.checked = 600000$
NL : Initially an area has to be continuously empty for at least 10 minutes before the lights in an area are switched off.

Property I_LCPFM_2

Formal : $lightOn.checked = 0$
NL : Initially the lights in an area are not switched off by the facility manager.

End of Initialization

Description Class RoomLightCtrlPanelFM

Intention

This class provides all the definitions of entities concerning the control of the light in a room for which the facility manager can enter a value using the control panel. In addition to the definitions of description class *AreaLightCtrlPanelFM* this is

- a possibility to determine the maximal delay within the reoccupation of a room leads to a reestablishing of the previous light scene.

Base Classes

Class *AreaLightCtrlPanelFM*

Intention : All definitions of description class *AreaLightCtrlPanelRoom* are also provided here.

Signature

Object $reoccDelay : EnteredValue(CHECKED\_DOM = TIME)$

Intention : This object represents the value a user can specify for the maximal delay within the reoccupation of a room leads to a reestablishing of the previous light scene.

Object $defaultLightScene : EnteredValue(CHECKED\_DOM = LightScene)$

Intention : This object represents the default light scene the facility manager can determine for a room.

Object $defaultAmbientLightLevel : EnteredValue(CHECKED\_DOM = LIGHT\_VALUES)$

Intention : This object represents the default ambient light level the facility manager can determine for a room.

Requirements/Specifications

Initialization

Property I_LCPR_1

Formal : $reoccDelay.checked = 1200000$
NL : Initially the maximal delay within the reoccupation of a room leads to a reestablishing of the previous light scene is 20 minutes.

End of Initialization

Property P_LCPR_1

Formal : $\Box(reoccDelay.checked > unoccDelay.checked)$
NL : The maximal delay within the reoccupation of a room leads to a reestablishing of the previous light scene is always greater than the time a room has to be continuously empty before the lights in a room are switched off.

Description Class LightCtrlPanelRoom

Intention

This class provides all the definitions of entities concerning the control of the light in a room for which a user can enter a value using the control panel of a room. These are

- a possibility to choose the current light scene
- a possibility to choose the current ambient light level
- a possibility to switch on and off the window ceiling light
- a possibility to switch on and off the wall ceiling light
- a possibility to determine whether
  - the currently set ambient light level should be applied or
  - the currently settings of the window and wall ceiling light switches of the control panel should be applied
- a possibility to determine whether
  - the light scenes are enabled by the control panel, i.e. the currently chosen light scene should be applied or
  - the currently settings of the control panel concerning the ambient switch, the ceiling light switches, and the ambient light level should be applied.
- a possibility to determine a default light scene
- a possibility to determine a default ambient light level

If the light is controlled using the control panel the following cases are possible:

- A chosen light scene has to be established. This is the case if the value of *lightSceneSwitch* is 1.
- A set ambient light level has to be established. This is the case if the value of *lightSceneSwitch* is 0 and the value of *ambientSwitch* is 1.
- The two lights are set directly using the control panel. This is the case if the value of *lightSceneSwitch* is 0 and the value of *ambientSwitch* is 0.

These different cases are specified in the description class *RoomLight*.

Signature

Object *chosenLightScene* : *EnteredValue*(*CHECKED_DOM = STRING*)

Intention : This object represents the light scene currently chosen by using the control panel. A light scene is identified by its name. Therefore, the entered value has to have the domain *STRING*.

Object *ambLightLev* : *EnteredValue*(*CHECKED_DOM = LIGHT_VALUES*)

Intention : This object represents the value defined for the ambient light level by using the control panel. The entered value has to be an element of the domain LIGHT_VALUES.
Note that this setting is only relevant if the light is not controlled manually using the push buttons.

Object *winCeilLightSwitch* : *EnteredValue*(*CHECKED_DOM = BINARY*)

Intention : This object represents the value defined for the window ceiling light by using the control panel.
winCeilLightSwitch.checked = 1 means that the window ceiling light should be on.
winCeilLightSwitch.checked = 0 means that the window ceiling light should be off.
Note that this setting is only relevant if the light is not controlled manually using the push buttons.

Object *wallCeilLightSwitch* : *EnteredValue*(*CHECKED_DOM = BINARY*)

Intention : This object represents the value defined for the wall ceiling light by using the control panel.
wallCeilLightSwitch.checked = 1 means that the wall ceiling light should be on.
wallCeilLightSwitch.checked = 0 means that the wall ceiling light should be off.
Note that this setting is only relevant if the light is not controlled manually using the push buttons.

Object *ambientSwitch* : *EnteredValue*(*CHECKED_DOM = BINARY*)

Intention : This object represents the value that indicates whether

- the currently set ambient light level (*ambLightLev*) should be applied (this is the case for *ambientSwitch.checked* = 1) or
- the currently settings of the window and wall ceiling light switches of the control panel should be applied (this is the case for *ambientSwitch.checked* = 0).

Note that this setting is only relevant if the light is not controlled manually using the push buttons.

Object *lightSceneSwitch* : *EnteredValue*(*CHECKED_DOM* = *BINARY*)

Intention : This object represents the value that indicates whether

- the light scenes are enabled by the control panel, i.e. the currently chosen light scene should be applied (this is the case for *lightSceneSwitch* = 1) or
- the currently settings of the control panel concerning the ambient switch (*ambientSwitch*), the ceiling light switches (*winCeilLightSwitch*, *winCeilLightSwitch*), and the ambient light level (*ambLightLev*) should be applied (this is the case for *lightSceneSwitch* = 0).

Note that this setting is only relevant if the light is not controlled manually using the push buttons.

Object *defaultLightScene* : *EnteredValue*(*CHECKED_DOM* = *STRING*)

Intention : This object represents the default light scene set by the user of a room.

Object *defaultAmbLightLevel* : *EnteredValue*(*CHECKED_DOM* = *LIGHT_VALUES*)

Intention : This object represents the default ambient light level set by the user of a room.

Description Class LightCtrlPanelBlindedRoom

Intention

This class provides all the definitions of entities concerning the control of the light in a blinded room for which a user can enter a value using the control panel.

In addition to the definitions of description class *LightCtrlPanelRoom* this class provides the definitions concerning the settings for sun blinds, i.e. for each sun blind in the room the control panel contains

- a switch to signal that a sun blind is controlled manually and
- a switch to determine the setting of a sun blind if it is controlled manually.

Formal Parameters

Function *numSunBlinds*() $\rightarrow$ *NAT*

Intention : This function represents the number of sun blinds that belong to a room and which can be controlled by the machine.

Base Classes

Class *LightCtrlPanelRoom*

Intention : All definitions of description class *LightCtrlPanelRoom* are also provided here.

Signature

Object *sunBlindSwitch* : *EnteredValue*(*CHECKED_DOM* = *THREE_PT_ACT_RANGE*)[*numSunBlinds*]

Intention : Each object represents the value set by the user for the control of a sun blind by using the control panel.

Object *sunBlindManual* : *EnteredValue*(*CHECKED_DOM* = *BINARY*)[*numSunBlinds*]

Intention : Each object represents whether the corresponding sun blind is controlled manually or not. A 1 means that it is controlled manually; a 0 means that it is not controlled manually.

Description Class TemperatureCtrlPanelFM

Intention

This class provides all the definitions of entities concerning the control of the temperature in a room for which the facility manager can enter a value using his control panel and which can not be changed by a user entering a value at the control panel in a room.

These objects are

- a possibility to determine the heating period. This object is the same for all rooms.

Values for which the facility manager specifies a default value that can be changed by a user are considered in the description class *TemperatureCtrlPanel*.

Signature

Object $heatingPeriod : CalendarInterval$

Intention : This object represents the heating period, i.e. the start and end point of the period and whether the actual date is within the heating period or not.
This object is the same for all rooms.

Description Class TemperatureCtrlPanelRoom

Intention

This class provides all the definitions of entities concerning the control of the temperature in a room for which a user can enter a value using the control panel of a room and for which the facility manager can not enter a default value. This is

- a possibility to determine the comfort time
- a possibility to determine the standby time
- a possibility to determine the holiday time
- a possibility to determine the working days

Values for which the facility manager specifies a default value that can be changed by a user are considered in the description class *TemperatureCtrlPanel*.

Signature

Object $comfortTime : DaytimeInterval$

Intention : This object represents the current comfort time.

Object $standbyTime : DaytimeInterval$

Intention : This object represents the current standby time.

Object $holidayTime : CalendarInterval$

Intention : This object represents the current holiday time.

Object $workingDays : WeekdayInterval$

Intention : This object represents the current working days.

Requirements/Specifications

Property P_TCPR_1

Formal : $\Box(\neg(standbyTime.actual \wedge comfortTime.actual))$

NL : The current daytime is never in the standby-time-interval and in the comfort-time-interval; i.e. the two daytime-intervals have to be disjoint.

Description Class TemperatureCtrlPanel

Intention

This class combines the two classes *TemperatureCtrlPanelFM* and *TemperatureCtrlPanelRoom* resulting in a class that contains all entities concerning the control of the temperature in a room for which a value can be entered using a control panel. Besides the collection of the entities of the two mentioned classes in this class also those entities are introduced for which the facility manager can specify a default value that can be changed by a user.

This class does not contain the values that can be entered concerning the malfunction of a sensor or actuator. They are introduced in the description classes *Sensors* and *Actuators*.

Signature

Object *tCPFM* : *TemperatureCtrlPanelFM*

Intention : This object represents the part of the temperature control panel containing the entities for which the facility manager can enter a value.

Object *tCPRoom* : *TemperatureCtrlPanelRoom*

Intention : This object represents the part of the temperature control panel containing the entities for which a user can enter a value.

Object *nonHeatingOccTimes* : *OccDelayTimes*

Intention : This object represents the set of delay times *entryTime*, dt_2, *exitTime* for the temperature operation mode *NON_HEATING*.
This object is specific for each room.

Object *offOccTimes* : *OccDelayTimes*

Intention : This object represents the set of delay times *entryTime*, dt_2, *exitTime* for the temperature operation mode *OFF*.
This object is specific for each room.

Object *standbyOccTimes* : *OccDelayTimes*

Intention : This object represents the set of delay times *entryTime*, dt_2, *exitTime* for the temperature operation mode *STANDBY*.
This object is specific for each room.

Object *comfortOccTimes* : *OccDelayTimes*

Intention : This object represents the set of delay times *entryTime*, dt_2, *exitTime* for the temperature operation mode *COMFORT*.
This object is specific for each room.

Object *comfortTemperature* : *ControlledInput*($INPUT\_DOM = TEMP$)

Intention : This object represents the desired temperature of a room in the case that the room is occupied or that the temperature operation mode *COMFORT* is active. The desired temperature depends on the values entered by the facility manager and the room user.
This object is specific for each room.

Object *offTemperature* : *ControlledInput*($INPUT\_DOM = TEMP$)

Intention : This object represents the desired temperature of a room in the case that the room is not occupied and the temperature operation mode *OFF* is active. The desired temperature depends on the values entered by the facility manager and the room user.
This object is specific for each room.

Object *standbyTemperature* : *ControlledInput*($INPUT\_DOM = TEMP$)

Intention : This object represents the desired temperature of a room in the case that the room is not occupied and the temperature operation mode *STANDBY* is active. The desired temperature depends on the values entered by the facility manager and the room user.
This object is specific for each room.

Object *heatUpStandby* : *ControlledInput*($INPUT\_DOM = TIME$)

Intention : This object represents the maximal time that should be needed to reach comfort temperature in the temperature operation mode *STANDBY*.

Object *heatUpNormal* : *ControlledInput*($INPUT\_DOM = TIME$)

Intention : This object represents the maximal time that should be needed to reach comfort temperature in the temperature operation mode *OFF*.

Requirements/Specifications

Property P_TCP_1

Formal : $\Box(heatUpNormal.value < heatUpStandby.value + 300000)$
NL : The time needed to reach comfort temperature from the temperature operation mode *STANDBY* is always at least 5 minutes greater than the time needed to reach comfort temperature from the temperature operation mode *OFF*.

Property P_TCP_2

NonFormal : The control panels should be easy and intuitive to use.

Group Occupancy

Intention

This group contains all description classes that contain definitions needed to define under which conditions an area is occupied.

Description Class Occupancy

Intention

This class provides all definitions to describe the term *occupancy* for an area. An area is here the generalization of rooms and hallways.

Whether the machine assumes occupancy in an area or not depends on

- three time values, dt_1, dt_2, dt_3, and
- on a predicate indicating whether a person is currently in the area or not.

Note that the status of *occupancy* in general does not correspond with the fact whether there is a person in an area or not. *Occupancy* is a derived term depending on this fact.

Signature

Timed Function $dt_1() \rightarrow TIME$

Intention	:	This function represents the current value used for the first time delay used to determine occupancy. The precise correlation between this time delay and *occupancy* is given in the property P_NMO_1.
Scope	:	machine controlled, not visible for the environment

Timed Function $dt_2() \rightarrow TIME$

Intention	:	This function represents the current value used for the second time delay used to determine occupancy. The precise correlation between this time delay and *occupancy* is given in property P_NMO_1.
Scope	:	machine controlled, not visible for the environment

Timed Function $dt_3() \rightarrow TIME$

Intention	:	This function represents the current value used for the third time delay used to determine occupancy. The precise correlation between this time delay and *occupancy* is given in property P_NMO_1.
Scope	:	machine controlled, not visible for the environment

Timed Predicate *emptyMach*

Intention	:	This predicate is true iff the machine assumes that there is currently no person in the area for which occupancy is considered.
Scope	:	machine controlled, not visible for the environment

Timed Predicate *occupied*

Intention : This predicate is true iff the machine assumes that the area for which occupancy is considered is occupied.
This predicate represents the situation as it is seen by the machine and not how it is in the real world.
Note that this predicate does not describe whether a person is in a area or not. See timed predicate *emptyMach*.

Scope : machine controlled, not visible for the environment

Requirements/Specifications

Property P_NMO_1

Formal : $\Box(occupied \leftrightarrow$
$\blacksquare_{\leq dt_1} \neg emptyMach \vee$
$< *[\blacksquare_{\leq dt_1 + dt_2} \neg emptyMach] \Rightarrow> \blacksquare\blacklozenge_{\leq dt_3} \neg emptyMach)$

NL : An area is defined to be occupied if and only if one of the following conditions holds:

- during the past dt_1 time units a person has continuously been in the area or
- there has already been a time span of length $dt_1 + dt_2$ such that a person has continuously been in the area, and since this has been the case for the last time, periods where the area has continuously been empty were not longer than or equal to dt_3.

Group BuildingUnits

Intention

This group contains all description classes that describe units of the floor considered in this problem specification.

Description Class AreaLight

Intention

This class provides all definitions to describe the general aspects of lights in an area. An area is here the generalization of rooms and hallways. In more detail this class provides:

- A set of lights that can be conrolled by the machine. The concrete number of controlled lights can be specified by instantiating the parameter *numLights*.
- The specification of the behavior of the system concerning the aspect *safety*. Here safety means that there is sufficient light to move. This class provides the requirements concerning the case that safe light has to be established. Other cases can be considered in derived classes.
- The specification of the behavior of the system concerning the case that the facility manager controls the light using a control panel.
- The specification of the behavior of the system concerning the case that the light is controlled manually.

Formal Parameters

Function $numLights() \rightarrow NAT$

Intention : This object represents the number of lights in an area that are controlled by the machine.

Signature

Object *occ* : *Occupancy*

Intention : This object represents the part of the definition of area light defining whether a room is occupied or not.

Object *light* : *ControlledLight*[*numLights*]

Intention : These objects represent the lights in area that are controlled by the machine.

Object *lCPFM* : *AreaLightCtrlPanelFM*

Intention : This object represents the light control panel containing the entities for which the facility manager can enter a value.

Time Constant T_AL_1

Intention : This time constant represents the maximal time delay for the following two situations:

- between the time point an area becomes occupied and the time point at which safe light is established;
- between the time point an area becomes unoccupied and the time point the machine controlled lights are off.

Scope : environment controlled, not visible for the machine

Time Constant T_AL_2

Intention : This time constant represents the maximal delay between the time point the machine assumes an occupation of an area and the time point the machine has set the corresponding actuators to establish safe light.

Scope : environment controlled, not visible for the machine

Timed Predicate *lightFMControlled*

Intention : This predicate is true iff the light is controlled by the facility manager using the control panel.

Scope : environment controlled, not visible for the machine

Timed Predicate *allLightsOff*

Intention : This predicate is true iff all lights in an area that are controlled by the machine are off.

Scope : environment controlled, not visible for the machine

Timed Predicate: *safeLight*

Intention : This predicate is true, iff there is sufficient light to move safely.

Scope : environment controlled, not visible for the machine

Timed Predicate *noSaveLight*

Intention : This predicate is true iff the light intensity in an area that is occupied can be less than the light intensity needed to establish *safe light*. This is for example the case if the light is controlled manually.

Scope : machine controlled, not visible for the environment

Timed Predicate *manualControl*

Intention : This predicate is true iff the machine assumes that the light in this area is controlled manually.

Scope : machine controlled, not visible for the environment

Function $evalSaveLightNAT(numLights) \to BINARY$

Intention	: This function represents the computation that results in the decision which light has to be on in order to achieve safe light in an area. $evalSaveLight$[i]=i means that the i–th light has to be on.
Scope	: machine controlled, not visible for the environment

Domain Knowledge

Property D_AL_1

Formal	: $\Box(T\_AL_2 < T\_AL_1)$
NL	: The time constant T_AL_2 is allways smaller than the time constant T_AL_1.

Property D_AL_2

Formal	: $\Box(allLightsOff \leftrightarrow \forall n \in NAT(numLights) : light[n].ls.noMalSens.envEntity = 0)$
NL	: The predicate $allLightsOff$ is true iff all lights that are controlled by the machine are off.

Requirements/Specifications

Initialization

Property I_AL_1

Formal	: $\neg noSaveLight$
NL	: Initially safe light has to be established if an area is occupied.

Property I_AL_2

Formal	: $\neg lightFMControlled$
NL	: Initially the light is not controlled by the facility manager using his control panel.

End of Initialization

Property P_AL_1

Formal	: $\Box(T\_AL_1 = 50)$
NL	: The value of T_AL_1 is always 50 miliiseconds.

Property P_AL_2

Formal	: $\Box(T\_AL_2 = 30)$
NL	: The value of T_AL_2 is always 30 miliiseconds.

Property P_AL_3

Formal	: $\Box(occ.dt_1 = 1000)$
NL	: The first time delay used to determine occupancy concerning the light control is always 1 second.

Property P_AL_4

Formal	: $\Box(occ.dt_2 = 0)$
NL	: The second time delay used to determine occupancy concerning the light control is always 0 second.

Property P_AL_5

Formal	: $\Box(occ.dt_3 = lCPFM.unoccDelay.checked)$
NL	: The third time delay used to determine occupancy concerning the light control corresponds always with the value entered by the facility manager for the time a certain area has to be unoccupied continuously before the lights are switched off.

Property P_AL_6

Formal	: $\Box(\neg noSaveLight \wedge occ.occupied \Rightarrow_{\leq T\_AL_1} safeLight)$
NL	: Whenever safe light has to be established and the area is occupied for at least T_AL_1 milliseconds, the light is sufficient to move safely within this time span and remains sufficient to move safely at least as long as the precondition is true.

Property P_AL_7

Formal	: $\Box(\neg noSaveLight \wedge occ.occupied \to \forall n \in NAT(numLights) : (evalSaveLight(n) = 1 \wedge light[n].ls.usedValue = 0 \to \Diamond_{\leq T\_AL_2} light[n].pulse.noMalAct.adjustedEntity = 1))$
NL	: Whenever safe light has to be established and the area is occupied the necessary actuators, i.e. pulse lines, are set within T_AL_2 milliseconds to establish safe light.

Property P_AL_8

Formal	: $\Box(lightFMControlled \wedge \neg occ.occupied \Rightarrow_{\leq T\_AL_1} \forall n \in NAT(numLights) : (light[n].ls.noMalSens.envEntity = lCPFM.lightOn.checked))$
NL	: Whenever the light is controlled by the facility manager using the control panel and the area is unoccupied for at least T_AL_1 milliseconds, the status of the machine controlled lights equals within this time the status as it is determined by the facility manager.

Property P_AL_9

Formal	: $\Box(\neg lightFMControlled \wedge \neg occ.occupied \Rightarrow_{\leq T\_AL_1} \forall n \in NAT(numLights) : (light[n].ls.noMalSens.envEntity = 0))$
NL	: Whenever the light is not controlled by the facility manager using the control panel and the area is unoccupied for at least T_AL_1 milliseconds, all machine controlled lights are off within this time and remains off at least as long as the precondition is true.

Property P_AL_{10}

Formal : $\Box(manualControl \leftrightarrow$
$\exists n \in NAT(numLights) : (light[n].manualControl) \wedge$
$occ.occupied)$

NL : The machine assumes that the light is controlled manually in an area iff if at least one light in this area is controlled manually and the area is occupied.

Property P_AL_{11}

Formal : $\Box(manualControl$
$\rightarrow \forall n \in NAT(numLights) :$
$(\neg[light[n].pulse.noMalAct.adjustedEntity = 1]))$

NL : If the light in an area is controlled manually, the settings of all lights remain unchanged.

Description Class HallwaySection

Intention

This class provides all definitions to describe the hallway specific light requirements. Each hallway section is equipped with

- one group of not dimmable ceiling lights that are considered here as one ceiling light,
- three motion detectors: one at each end of a hallway section and one in the middle of a hallway section.

Note that there are no requirements concerning the temperature in a hallway.

Base Classes

Class $AreaLight(numLights = 1)$

Intention : All definitions of description class *AreaLight* are also provided here. There is one light in a hallway section that is controlled by the machine.

Signature

Object $eastMd : MotionDetector$

Intention : This object represents the motion detector in a hallway which is installed at the more eastern door of the hallway.

Object $westMd : MotionDetector$

Intention : This object represents the motion detector in a hallway which is installed at the more western door of the hallway.

Object $middleMd : MotionDetector$

Intention : This object represents the motion detector in a hallway which is installed in the middle of the hallway.

Time Constant T_HS_1

Intention : This constant represents the time delay between a time point a person is near a hallway door indicating that he wants to enter this hallway section and switching on the lights.

Scope : environment controlled, not visible for the machine

Timed Predicate $nearHallwayDoor$

Intention : This predicate is true iff a person is near a hallway door to this hallway section indicating that he wants to enter the hallway section.

Scope : environment controlled, not visible for the machine

Requirements/Specifications

Property P_HS_1

Formal	: $\Box(T\_HS_1 = 50)$
NL	: The value of T_HS_1 is always 50 miliiseconds.

Property P_HS_2

Formal	: $\Box(\neg manualControl \rightarrow \neg noSaveLight)$
NL	: Whenever the light is not controlled manually the light intensity in a hallway section that is occupied has to be greater than or equal to the light intensity needed to establish *save light*.

Property P_HS_3

Formal	: $\Box(nearHallwayDoor \Rightarrow_{\leq T\_HS_1} light[1].ls.noMalSens.envEntity = 1)$
NL	: Whenever a person is continuously in the near of a hallway door of the hallway section for at least T_HS_1 time units indicating that he wants to enter the hallway section, then the light in the hallway section will be on within this time span, and remains on at least as long as the person is near the hallway section.

Property P_HS_4

Formal	: $\Box(occ.emptyMach \leftrightarrow ((eastMd.usedValue = 0) \wedge (westMd.usedValue = 0) \wedge (middleMd.usedValue = 0)))$
NL	: The machine always assumes that there is currently no person in a hallway section iff the values of all three motion detectors are 0.

Description Class RoomLight

Intention

This class provides all definitions to describe the environment and the requirements concerning the control of lights common to any kind of room. In more detail this means that the cases are considered

- that safe light has to be established. This behavior is inherited from the description class *AreaLight*;
- that the light is controlled manually.
- that the light is controlled using the control panel. See the intention of description class *LightCtrlPanelRoom* for a more detailed explanation of this case.

In each room there are two dimmable lights that are controlled by the machine: a ceiling light at the window and a ceiling light at the wall to the hallway. Furthermore, in each room there is a control panel for entering values relevant for the control of the light. This is done by the user of a room. This control panel is not the same as the one the facility manager has.

Base Classes

Class $AreaLight(numLights = 2)$

Intention	: All definitions of description class Area are also provided here. There are two lights in a room that are controlled by the machine: $light[1]$ represents the light near the window and $light[2]$ is the light near the wall to the hallway.

Signature

Object $AreaLight :: light : DimmableControlledLight$

Intention	: The lights in a room are *dimmable* lights.

Object $AreaLight :: lCPFM : RoomLightCtrlPanelFM$

Intention	: The control panel of the facility manager in a room provides some further input possibilities.

Object $lCPRoom : LightCtrlPanelRoom$

Intention	: This object represents the light control panel containing the entities for which a user can enter a value.

Time Constant T_RL_1

Intention	: This time constant represents the maximal delay between occupation of a room, when a light scene is chosen, and the establishment of the chosen light scene.
Scope	: environment controlled, not visible for the machine

Timed Function $curLightScene() \rightarrow LightScene$

Intention : This function represents the current light scene that is established by the machine if light scenes are enabled.
Scope : machine controlled, not visible for the environment

Timed Function $curStandLightScene() \rightarrow LightScene$

Intention : This function represents the current standard (=default) light scene that is established by the machine, if light scenes are enabled.
Scope : machine controlled, not visible for the environment

Timed Function $curAmbLightLev() \rightarrow LIGHT\_VALUES$

Intention : This function represents the current ambient light level that is established by the machine if ambient light settings are enabled by the control panel.
Scope : machine controlled, not visible for the environment

Timed Function $defaultAmbLightLev() \rightarrow LIGHT\_VALUES$

Intention : This function represents the default ambient light level.
Scope : machine controlled, not visible for the environment

Timed Function $determineLightScene(STRING) \rightarrow LightScene$

Intention : This function determines the light scene which is identified by the given name.
Scope : machine controlled, not visible for the environment

Function $evalLightScene(LightScene, LightScene :: LAMPS) \rightarrow PERCENT$

Intention : This function represents the computation of the dimmer settings for a given lighting (represented by the second argument) that are necessary to achieve the ambient light intensity of the given light scene (represented by the first argument) under consideration of the light scene's lighting order. The value of this function represents the dimmer settings.
Scope : machine controlled, not visible for the environment

Function $evalAmbient(LIGHT\_VALUES, LightScene :: LAMPS) \rightarrow PERCENT$

Intention : This function represents the dimmer setting for a given lighting (represented by the second argument) that is necessary to achieve the given ambient light intensity (represented by the first argument) together with the other lightings.
Scope : machine controlled, not visible for the environment

Requirements/Specifications

Initialization

Property I_RL_1

Formal : $curLightScene = curStandLightScene$
NL : Initially the current light scene is the current standard light scene.

Property I_RL_2

Formal : $curAmbLightLev = defaultAmbLightLev$
NL : Initially the current ambient light level is the current default ambient light level.

End of Initialization

Property P_RL_1

Formal : $\Box(T\_RL_1 = 50)$
NL : The value of T_RL_1 is always 50 miliiseconds.

Property P_RL_2

Formal : $\Box(curStandLightScene = lCPFM.defaultLightScene.checked)$
NL : The current standard light scene always corresponds with the default light scene determined by the facility manager.

Property P_RL_3

Formal : $\Box(curAmbLightLev = lCPFM.defaultAmbientLightLevel.checked)$
NL : The current ambient light level always corresponds with the default ambient light level determined by the facility manager.

Property P_RL_4

Formal : $\Box(noSaveLight \leftrightarrow manualControl \vee lCPRoom.lightSceneSwitch.checked = 1 \vee lCPRoom.ambientSwitch.checked = 1)$
NL : The light intensity in a room that is occupied can be less than the light intensity needed to establish *save light* iff the light is controlled manually or light scenes are enabled or the ambient light level is enabled.

Property P_RL_5

Formal : $\Box(([occ.occupied] \wedge \blacklozenge_{\leq lCPFM.reoccDelay.checked} occ.occupied) \rightarrow curLightScene = determineLightScene(lCPRoom.chosenLightScene.checked))$
NL : Whenever the room becomes occupied now and was occupied sometimes within the last $lCPFM.reoccDelay.checked$ milliseconds and the light scene l was last chosen, the light scene l is the current light scene.

Property P_RL_6

Formal : $\Box(([occ.occupied] \wedge \blacksquare_{> lCPFM.reoccDelay.checked} \neg occ.occupied) \rightarrow curLightScene = curStandLightScene)$
NL : Whenever the room becomes occupied now and has not been occupied during the last $lCPFM.reoccDelay.checked$ milliseconds, the current light scene is the current standard light scene.

Property P_RL_7

Formal :
$$\Box(\forall l \in LightScene : ((occ.occupied \wedge \neg[occ.occupied] \wedge [determineLightScene(lCPRoom.chosenLightScene.checked) = l]) \rightarrow \Diamond_{\leq T\_RL_1} curLightScene = l))$$

NL : Whenever the room is occupied, but has not just become occupied now, and the light scene l becomes chosen, this light scene l is the current light scene within time T_RL_1.

Property P_RL_8

Formal :
$$\begin{aligned}\Box((&occ.occupied \wedge \neg manualControl \wedge lCPRoom.lightSceneSwitch.checked = 1)\\ \rightarrow ((&light[1].pulse.noMalAct.adjustedEntity = 1 \leftrightarrow \neg evalLightScene(curLightScene, LightScene :: WINDOW) = 0) \wedge\\ &(light[1].dimmer.noMalAct.adjustedEntity = evalLightScene(curLightScene, LightScene :: WINDOW)) \wedge\\ &(light[2].pulse.noMalAct.adjustedEntity = 1 \leftrightarrow \neg evalLightScene(curLightScene, LightScene :: WALL) = 0) \wedge\\ &(light[2].dimmer.noMalAct.adjustedEntity = evalLightScene(curLightScene, LightScene :: WALL))))\end{aligned}$$

NL : Whenever the room is occupied and the light is not manually controlled and light scenes are enabled by the control panel, the window and wall lights are dimmed and switched on or off as necessary to achieve the ambient light level of the current light scene under consideration of the current light scene's lighting order.
In short: Under the mentioned conditions the current light scene has to be established.

Property P_RL_9

Formal :
$$\begin{aligned}\Box((&occ.occupied \wedge \neg manualControl \wedge lCPRoom.lightSceneSwitch.checked = 0 \wedge lCPRoom.ambientSwitch.checked = 1)\\ \rightarrow ((&light[1].pulse.noMalAct.adjustedEntity = 1 \leftrightarrow \neg evalAmbient(curAmbLightLev, LightScene :: WINDOW) = 0) \wedge\\ &(light[1].dimmer.noMalAct.adjustedEntity = evalAmbient(curAmbLightLev, LightScene :: WINDOW)) \wedge\\ &(light[2].pulse.noMalAct.adjustedEntity = 1 \leftrightarrow \neg evalAmbient(curAmbLightLev, LightScene :: WALL) = 0) \wedge\\ &(light[2].dimmer.noMalAct.adjustedEntity = evalAmbient(curAmbLightLev, LightScene :: WALL))))\end{aligned}$$

NL : Whenever the room is occupied and the light is not manually controlled, light scenes are disabled and ambient light settings are enabled by the control panel, the window and wall lights are dimmed and switched on or off as necessary to achieve the current ambient light level.
In short: Under the mentioned conditions the current ambient light setting is established.

Property P_RL_{10}

Formal :
$$\begin{aligned}\Box(&occ.occupied \wedge \neg manualControl \wedge lCPRoom.lightSceneSwitch.checked = 0 \wedge lCPRoom.ambientSwitch.checked = 0\\ \rightarrow ((&light[1].pulse.noMalAct.adjustedEntity = lCPRoom.winCeilLightSwitch.checked) \wedge\\ &(lCPRoom.winCeilLightSwitch.checked = 1 \rightarrow light[1].dimmer.noMalAct.adjustedEntity = 100) \wedge\\ &(lCPRoom.winCeilLightSwitch.checked = 0 \rightarrow light[1].dimmer.noMalAct.adjustedEntity = 0) \wedge\\ &(light[2].pulse.noMalAct.adjustedEntity = lCPRoom.wallCeilLightSwitch.checked) \wedge\\ &(lCPRoom.wallCeilLightSwitch.checked = 1 \rightarrow light[2].dimmer.noMalAct.adjustedEntity = 100) \wedge\\ &(lCPRoom.wallCeilLightSwitch.checked = 0 \rightarrow light[2].dimmer.noMalAct.adjustedEntity = 0)))\end{aligned}$$

NL : Whenever the room is occupied and the light is not manually controlled and light scenes and ambient light setting are disabled by the control panel, the window and wall lights are completely switched off or on as selected in the control panels ceiling light switches.
In short: Under the mentioned conditions the light is controlled by the ceiling light switches.

Property P_RL_{11}

Formal : $\Box(manualControl$
$\rightarrow \forall p \in PERCENT:$
$(light[1].dimmer.noMalAct.adjustedEntity = p$
$\rightarrow (light[1].dimmer.noMalAct.adjustedEntity = p$
$\mathcal{W} \neg manualControl)) \wedge$
$\forall p \in PERCENT:$
$(light[2].dimmer.noMalAct.adjustedEntity = p$
$\rightarrow (light[2].dimmer.noMalAct.adjustedEntity = p$
$\mathcal{W} \neg manualControl)))$

NL : As long as the light is controlled manually, the dimmer settings of the window and wall light remain unchanged.

Property P_RL_{12}

Formal : $\Box(([occ.occupied] \wedge \blacklozenge_{\leq lCPFM.reoccDelay.checked} occ.occupied)$
$\rightarrow curAmbLightLev = lCPRoom.ambLightLev.checked)$

NL : Whenever the room becomes occupied now and was occupied sometimes within the last *lCPFM.reoccDelay.checked* milliseconds and the ambient light level l was last chosen, the ambient light level l is the current ambient light level.

Property P_RL_{13}

Formal : $\Box(([occ.occupied] \wedge \blacksquare_{> lCPFM.reoccDelay.checked} \neg occ.occupied)$
$\rightarrow curAmbLightLev = defaultAmbLightLev)$

NL : Whenever the room becomes occupied now and has not been occupied during the last *lCPFM.reoccDelay.checked* milliseconds, the current ambient light level is the default ambient light level.

Property P_RL_{14}

Formal : $\Box(\forall l \in LIGHT\_VALUES:$
$((occ.occupied \wedge \neg[occ.occupied] \wedge$
$[lCPRoom.ambLightLev.checked = l])$
$\rightarrow \Diamond_{\leq T\_RL_1} curAmbLightLev = l))$

NL : Whenever the room is occupied, but has not just become occupied now, and the ambient light level l becomes chosen, this ambient light level l is the current ambient light level within time T_RL_1.

Description Class BlindedRoomLight

Intention

This class provides all definitions to describe the control of lights common to any kind of room having sun blinds. It extends the definitions of the description class *RoomLight*. The special equippment of such rooms includes:

- *numSunBlinds* many blinds
- a control panel including possibilities to determine the settings of the sun blinds

Formal Parameters

Function $numSunBlinds() \rightarrow NAT$

Intention : This function represents the number of sun blinds that belong to a room and which can be controlled by the machine.

Base Classes

Class *RoomLight*

Intention : All definitions of description class *RoomLight* are also provided here.

Signature

Object $RoomLight :: lCPRoom :$
$LightCtrlPanelBlindedRoom(numSunBlinds = numSunBlinds)$

Intention : This object represents the light specific inputs of a user including possibilities to determine the settings of the sun blinds.

Timed Function $lightBlindSet(NAT(numSunBlinds)) \rightarrow$
$THREE\_PT\_ACT\_RANGE$

Intention : This function represents the current settings of the sun blinds in a room intended by the light control.

Scope : machine controlled, not visible for the environment

Description Class RoomTemperature

Intention

This class provides all the definitions to describe the determination of the desired room temperature. For this the definitions of several objects are combined:

- an object representing the temperature operation modes, this object includes the occupancy determination for a room;
- an object representing the control panel that a person can use to enter values controlling the computation of the desired room temperature.

Note that in this class it is not specified how the desired room temperature is achieved.

Signature

Domain $TEMP\_OP\_MODES = \{NON\_HEATING, OFF, STANDBY, COMFORT\}$

Intention : This domain contains the possible temperature operation modes.

Object cp : *TemperatureCtrlPanel*

Intention : This object represents the temperature specific inputs of a user and the facility manager.

Object occ : *Occupancy*

Intention : This object represents the part of the definition defining whether a room is occupied or not.

Timed Function $currentTempOpMode() \rightarrow TEMP\_OP\_MODES$

Intention : This function represents the current temperature operation mode.
Scope : machine controlled, not visible for the environment

Timed Function $roomDesiredTemperature() \rightarrow TEMP$

Intention : This function represents the currently desired temperature that should be established in a room.
Scope : machine controlled, not visible for the environment

Requirements/Specifications

Property P_RT_1

Formal : $\Box(\neg cp.tCPFM.heatingPeriod.actual \rightarrow currentTempOpMode = NON\_HEATING)$

NL : Whenever the current day is not within the heating period the current temperature operation mode is *NON_HEATING*.

Property P_RT_2

Formal : $\Box(cp.tCPFM.heatingPeriod.actual \wedge cp.tCPRoom.holidayTime.actual \rightarrow currentTempOpMode = OFF)$

NL : Whenever the current day is within the heating period and within the holiday time the current temperature operation mode is *OFF*.

Property P_RT_3

Formal : $\Box(cp.tCPFM.heatingPeriod.actual \wedge \neg cp.tCPRoom.holidayTime.actual \wedge \neg cp.tCPRoom.workingDays.actual \rightarrow currentTempOpMode = OFF)$

NL : Whenever the current day is within the heating period and not within the holiday time and the current week-day is not a working day the temperature operation mode is *OFF*.

Property P_RT_4

Formal : $\Box(cp.tCPFM.heatingPeriod.actual \wedge \neg cp.tCPRoom.holidayTime.actual \wedge cp.tCPRoom.workingDays.actual \wedge \neg cp.tCPRoom.comfortTime.actual \wedge \neg cp.tCPRoom.standbyTime.actual \rightarrow currentTempOpMode = OFF)$

NL : Whenever the current day is within the heating period and not within the holiday time and the current week-day is a working day and the current daytime is neither within the comfort-time nor within the standby-time (i.e. it is within the off-time) the temperature operation mode is *OFF*.

Property P_RT_5

Formal : $\Box(cp.tCPFM.heatingPeriod.actual \wedge \neg cp.tCPRoom.holidayTime.actual \wedge cp.tCPRoom.workingDays.actual \wedge cp.tCPRoom.standbyTime.actual \rightarrow currentTempOpMode = STANDBY)$

NL : Whenever the current day is within the heating period and not within the holiday time and the current week-day is a working day and the current daytime is within the standby-time the temperature operation mode is *STANDBY*.

Property P_RT_6

Formal : $\Box(cp.tCPFM.heatingPeriod.actual \wedge$
$\neg cp.tCPRoom.holidayTime.actual \wedge$
$cp.tCPRoom.workingDays.actual \wedge$
$cp.tCPRoom.comfortTime.actual$
$\rightarrow currentTempOpMode = COMFORT)$

NL : Whenever the current day is within the heating period and not within the holiday time and the current week-day is a working day and the current daytime is within the comfort-time the temperature operation mode is *COMFORT*.

Property P_RT_7

Formal : $\Box(currentTempOpMode = COMFORT$
$\rightarrow roomDesiredTemperature = cp.comfortTemperature.value)$

NL : Whenever the current temperature operation mode is *COMFORT* the desired room temperature is the comfort temperature.

Property P_RT_8

Formal : $\Box(occ.occupied$
$\rightarrow roomDesiredTemperature = cp.comfortTemperature.value)$

NL : Whenever the room is occupied the desired room temperature is the comfort temperature.

Property P_RT_9

Formal : $\Box(currentTempOpMode = STANDBY \wedge \neg occ.occupied$
$\rightarrow roomDesiredTemperature = cp.standbyTemperature.value)$

NL : Whenever the current temperature operation mode is *STANDBY* and the room is not occupied the desired room temperature is stand-by temperature.

Property P_RT_{10}

Formal : $\Box(currentTempOpMode = OFF \wedge \neg occ.occupied$
$\rightarrow roomDesiredTemperature = cp.offTemperature.value)$

NL : Whenever the current temperature operation mode is *OFF* and the room is not occupied the desired room temperature is off temperature.

Property P_RT_{11}

Formal : $\Box(currentTempOpMode = NON\_HEATING \rightarrow$
$occ.dt_1 = cp.nonHeatingOccTimes.entryTime.value \wedge$
$occ.dt_2 = cp.nonHeatingOccTimes.dt_2.value \wedge$
$occ.dt_1 = cp.nonHeatingOccTimes.exitTime.value)$

NL : The time delays used for determining occupancy if the temperature operation mode *NON_HEATING* is active corresponds with those entered for this case.

Property P_RT_{12}

Formal : $\Box(currentTempOpMode = OFF \rightarrow$
$occ.dt_1 = cp.offOccTimes.entryTime.value \wedge$
$occ.dt_2 = cp.offOccTimes.dt_2.value \wedge$
$occ.dt_1 = cp.offOccTimes.exitTime.value)$

NL : The time delays used for determining occupancy if the temperature operation mode *OFF* is active corresponds with those entered for this case.

Property P_RT_{13}

Formal : $\Box(currentTempOpMode = STANDBY \rightarrow$
$occ.dt_1 = cp.standbyOccTimes.entryTime.value \wedge$
$occ.dt_2 = cp.standbyOccTimes.dt_2.value \wedge$
$occ.dt_1 = cp.standbyOccTimes.exitTime.value)$

NL : The time delays used for determining occupancy if the temperature operation mode *STANDBY* is active corresponds with those entered for this case.

Property P_RT_{14}

Formal : $\Box(currentTempOpMode = COMFORT \rightarrow$
$occ.dt_1 = cp.comfortOccTimes.entryTime.value \wedge$
$occ.dt_2 = cp.comfortOccTimes.dt_2.value \wedge$
$occ.dt_1 = cp.comfortOccTimes.exitTime.value)$

NL : The time delays used for determining occupancy if the temperature operation mode *COMFORT* is active corresponds with those entered for this case.

Property P_RT_{15}

NonFormal : The use of solar radiation for heating during all times if heating is required or not disallowed, and the use of storage capacity of the building should always be preferred against the usage of the central heating unit.

Description Class BlindedRoomTemperature

Intention

This class provides all definitions to describe the control of the temperature common to any kind of room having sun blinds. It extends the definitions of the description class *RoomTemperature*. The special equippment of such rooms includes:

- numSunBlinds many blinds

The current setting of the sun blinds in a room intended by the temperature control depends on the currently desired room temperature.

Formal Parameters

Function $numSunBlinds() \to NAT$

Intention	: This function represents the number of sun blinds that belong to a room and which can be controlled by the machine.

Base Classes

Class *RoomTemperature*

Intention	: All definitions of description class *RoomTemperature* are also provided here.

Signature

Timed Function $tempBlindSet(NAT(numSunBlinds), TEMP) \to THREE\_PT\_ACT\_RANGE$

Intention	: This function represents the current setting of the sun blinds in a room intended by the temperature control. These settings depend (at least) on the current desired room temperature (second argument).
Scope	: machine controlled, not visible for the environment

Description Class BasicRoom

Intention

This class provides all definitions to describe the behavior common to any kind of room. This includes the definitions of the light and temperature control in a room as well as the aggregation of the objects that are installed in each room. In this class we do not consider such objects for which the number of installed instances can vary from room to room, e.g. the number of installed radiators. These objects are integrated in the derived description class *Room*.

Each room is equipped with

- one motion detector
- three temperature sensors

Signature

Object $RL : RoomLight$

Intention	: This object represents the part that describes the behavior of a room concerning the light control.

Object $RT : RoomTemperature$

Intention	: This object represents the part that describes the behavior of a room concerning the temperature control.

Object $md : MotionDetector$

Intention	: This object represents the room motion detector.

Object $rt : RoomTempSensor[3]$

Intention	: Each object represents one of the three air temperature sensors in a room.

Requirements/Specifications

Property P_BaR_1

Formal	: $\Box(RL.occ.emptyMach \leftrightarrow md.usedValue = 0)$
NL	: Concerning the light control the machine always assumes that there is currently no person in a room iff the value of the motion detector is 0.

Property P_BaR_2

Formal	: $\Box(RT.occ.emptyMach \leftrightarrow md.usedValue = 0)$
NL	: Concerning the temperature control the machine always assumes that there is currently no person in a room iff the value of the motion detector is 0.

Description Class Room

Intention

This class extends the description class *BasicRoom* with the definitions concerning those objects of which in each room at least one instantiation is installed but where the concrete number of instantion can vary from room to room.

In addition to the objects introduced in description class *BasicRoom* each room is equipped with

- *numRad* many radiators
- *numDoorHallway* many doors to the hallway

Formal Parameters

Function $numRad() \to NAT$

Intention : This function represents the number of radiators that belong to a room.

Base Classes

Class *BasicRoom*

Intention : All definitions of description class *BasicRoom* are also provided here.

Signature

Object $rad : Radiator[numRad]$

Intention : Each object represents one radiator of a room.

Timed Function $tempRadSet(TEMP) \to THREE\_PT\_ACT\_RANGE$

Intention : This function represents the current setting of the radiators in a room intended by the temperature control.

Scope : machine controlled, not visible for the environment

Requirements/Specifications

Property P_R_1

Formal : $\Box(\forall n \in NAT(numRad) : rad[n].valveAct.noMalAct.adjustedEntity = tempRadSet(RT.roomDesiredTemperature))$

NL : The value adjusted by the machine for the radiator valves corresponds with the value intended by the temperature control for these actuators depending on the currently desired temperature.
Note that all radiator valves in a room are always set to the same value. A specific value for a single radiator is not determined.

Property P_R_2

NonFormal : The desired temperature shall be reached as fast as possible during the heating up state.
This requirement is a basis for the definition of the function *tempRadSet*.

Property P_R_3

NonFormal : The currently desired temperature is maintained during the current temperature state as good as possible.
This requirement is a basis for the definition of the function *tempRadSet*.

Description Class BlindedRoom

Intention

This class provides all definitions specific to any kind of room with sun blinds. In addition to the definitions of the description class *Room* the control of the sun blinds is considered here. Since both, light and temperature control, can influence the setting of the sun blinds there can be conflicts. The resolution of possible conflicts is represented by the function *resolveBlindConflict*.

Formal Parameters

Function $numRad() \rightarrow NAT$

Intention	: This function represents the number of radiators that belong to a room.

Function $numSunBlinds() \rightarrow NAT$

Intention	: This function represents the number of sun blinds that belong to a room and which can be controlled by the machine.

Base Classes

Class $Room(numRad = numRad)$

Intention	: All definitions of description class *Room* are also provided here.

Signature

Object $sb : SunBlind[numSunBlinds]$

Intention	: Each object represents one sun blind of a room that can be controlled by the machine.

Object $BasicRoom :: RL : BlindedRoomLight(numSunBlinds = numSunBlinds)$

Intention	: This object represents the light control in a blinded room.

Object $BasicRoom :: RT :$
$BlindedRoomTemperature(numSunBlinds = numSunBlinds)$

Intention	: This object represents the temperature control in a blinded room.

Timed Function $resolveBlindConflict(NAT(numSunBlinds), THREE\_PT\_ACT\_RANGE, THREE\_PT\_ACT\_RANGE) \rightarrow THREE\_PT\_ACT\_RANGE$

Intention	: This function represents the current settings of the sun blinds in a room intended by the machine. These settings are computed based on the intended settings of the temperature control (second argument) and the intended settings of the light control (third argument).
Scope	: machine controlled, not visible for the environment

Timed Predicate $manualBlind$

Intention	: This predicate is true iff the machine assumes that the sun blinds are controlled manually and not by the control system.
Scope	: machine controlled, not visible for the environment

Domain Knowledge

Property D_BR_1

Formal	: $\Box(numSunBlinds = numRad)$
NL	: The number of sun blinds in a blinded room equals the number of radiators in a blinded room.

Requirements/Specifications

Initialization

Property I_BR_1

Formal	: $\neg manualBlind$
NL	: Initially the sun blinds are not controlled manually.

End of Initialization

Property P_BR_1

Formal	: $\Box(manualBlind \leftrightarrow RT.occ.occupied \wedge \exists s \in NAT(numSunBlinds) : RL.lCPRoom.sunBlindManual[s].checked = 1)$
NL	: The machine assumes that the sun blinds are controlled manually iff a room is occupied according to the definition of this term used for temperature control and at least one sun blind is controlled manually.

Property P_BR_2

Formal	: $\Box(manualBlind \rightarrow \forall s \in NAT(numSunBlinds) : sb[s].blind.noMalAct.adjustedEntity = RL.lCPRoom.sunBlindSwitch[s].checked)$
NL	: If the sun blinds are controlled manually the value adjusted by the machine for each sun blind is equal to the corresponding switch status of the control panel.

Property P_BR_3

Formal	: $\Box(\neg manualBlind \rightarrow \forall s \in NAT(numSunBlinds) : sb[s].blind.noMalAct.adjustedEntity = resolveBlindConflict(s, RT.tempBlindSet(s, RT.roomDesiredTemperature), RL.lightBlindSet(s)))$
NL	: If the sun blinds are not controlled manually the value adjusted by the machine for each sun blind is equal to the value determined for the sun blinds by the function *resolveBlindConflict*.

Description Class StairCase

Intention

This class provides all definitions to describe the entities belonging to a staircase which are relevant for the considered system. This is only a motion detector.

Signature

Object $md : MotionDetector$

Intention : This object represents a motion detector placed in a staircase in front of the hallway door.

Main Description Class Floor

Intention

This class is the main class. It contains and combines all the objects belonging to the considered system. These objects are

- the rooms of the floor
- the hallway sections of the floor
- the parts of the staircases belonging to the floor
- the outdoor light sensors
- the outdoor temperature sensor

Signature

Object $o415 : Room(numRad = 2)$

Intention : This object represents the office with room number 415. This office is equipped with

- two radiators
- one door to the hallway

Object $o417 : Room(numRad = 2)$

Intention : This object represents the office with room number 417. This office is equipped with

- two radiators
- one door to the hallway

Object $o419 : Room(numRad = 2)$

Intention : This object represents the office with room number 419. This office is equipped with

- two radiators
- one door to the hallway

Object $o429$: $Room(numRad = 2)$

Intention : This object represents the office with room number 429. This office is equipped with

- two radiators
- one door to the hallway

Object $o425$: $Room(numRad = 2)$

Intention : This object represents the office with room number 425. This office is equipped with

- two radiators
- one door to the hallway
- one door to a neighbour room

Object $o431$: $Room(numRad = 2)$

Intention : This object represents the office with room number 431. This office is equipped with

- two radiators
- one door to the hallway
- one door to a neighbour room

Object $o435$: $Room(numRad = 2)$

Intention : This object represents the office with room number 435. This office is equipped with

- two radiators
- one door to the hallway
- one door to a neighbour room

Object $o423$: $Room(numRad = 2)$

Intention : This object represents the office with room number 423. This office is equipped with

- two radiators
- one door to the hallway
- two doors to neighbour rooms

Object $o433$: $Room(numRad = 2)$

Intention : This object represents the office with room number 433. This office is equipped with

- two radiators
- one door to the hallway
- two doors to neighbour rooms

Object $o421$: $Room(numRad = 3)$

Intention : This object represents the office with room number 421. This office is equipped with

- three radiators
- one door to the hallway
- one door to a neighbour room

Object $m427$: $Room(numRad = 5)$

Intention : This object represents the meeting room with room number 427. It is equipped with

- five radiators
- one door to the hallway

Object $cl411$: $Room(numRad = 9)$

Intention : This object represents the computer lab with room number 411. It is equipped with

- nine radiators
- two doors to the hallway

Object $cl426$: $BlindedRoom(numRad = 3,$
$numSunBlinds = 3)$

Intention : This object represents the computer lab with room number 426. It is equipped with

- three radiators
- three sun blinds
- one door to the hallway

Object $o424$: $BlindedRoom(numRad = 2,$
$numSunBlinds = 2)$

Intention : This object represents the office with room number 424. It is equipped with

- two radiators
- two sun blinds
- one door to the hallway

Object $cl422$: $BlindedRoom(numRad = 3,$
$numSunBlinds = 3)$

Intention : This object represents the computer lab with room number 424. It is equipped with

- three radiators
- three sun blinds
- one door to the hallway
- one door to a neighbour room

Object $cl418$: $BlindedRoom(numRad = 2,$
$numSunBlinds = 2)$

Intention : This object represents the computer lab with room number 418. It is equipped with

- two radiators
- two sun blinds
- one door to the hallway
- two doors to neighbour rooms

Object $o416$: $BlindedRoom(numRad = 2,$
$numSunBlinds = 2)$

Intention : This object represents the office with room number 416. It is equipped with

- two radiators
- two sun blinds
- one door to the hallway
- two doors to neighbour rooms

Object $o414$: $BlindedRoom(numRad = 2,$
$numSunBlinds = 2)$

Intention : This object represents the office with room number 414. It is equipped with

- two radiators
- two sun blinds
- one door to the hallway
- one door to a neighbour rooms

Object *o412* : $BlindedRoom(numRad = 3,$
$numSunBlinds = 3)$

Intention : This object represents the office with room number 412. It is equipped with

- three radiators
- three sun blinds
- one door to the hallway

Object *cl410* : $BlindedRoom(numRad = 10,$
$numSunBlinds = 10)$

Intention : This object represents the computer lab with room number 410. It is equipped with

- ten radiators
- ten sun blinds
- three doors to the hallway

Object *h1* : *HallwaySection*

Intention : This object represents the hallway in the east of floor 32/4.

Object *h2* : *HallwaySection*

Intention : This object represents the hallway in the middle of floor 32/4.

Object *h3* : *HallwaySection*

Intention : This object represents the hallway in the west of floor 32/4.

Object *ols1* : *OutdoorLightSensor*

Intention : This object represents the outdoor light sensor in the north-east direction.

Object *ols2* : *OutdoorLightSensor*

Intention : This object represents the outdoor light sensor in the south-west direction.

Object *ols3* : *OutdoorLightSensor*

Intention : This object represents the outdoor light sensor in the north-west direction.

Object *ols4* : *OutdoorLightSensor*

Intention : This object represents the outdoor light sensor in the south-east direction.

Object *ols5* : *OutdoorLightSensor*

Intention : This object represents the outdoor light sensor in the north direction.

Object *ols6* : *OutdoorLightSensor*

Intention : This object represents the outdoor light sensor in the south direction.

Object *ots* : *OutdoorTempSensor*

Intention : This object represents the outdoor temperature sensor.

Object *scw* : *StairCase*

Intention : This object represents the staircase in the west of floor 32/4.

Object *sce* : *StairCase*

Intention : This object represents the staircase in the east of floor 32/4.

Domain $ROOMS = \{o435, o433, o431, o429, o425, o423, o421, o419, o417, o415,$
$o424, o416, o414, o412, m427, cl411, cl426, cl422,$
$cl418, cl410\} < BasicRoom$

Intention : This domain contains all the rooms that have to be considered.

Domain $OFFICES = \{o435, o433, o431, o429, o425, o423, o421, o419, o417, o415,$
$o424, o416, o414, o412\} < ROOMS$

Intention : This domain contains all the offices that have to be considered.

Domain $MEETING\_ROOMS = \{m427\} < ROOMS$

Intention : This domain contains all the meeting rooms that have to be considered.

Domain $COMPUTER\_LABS = \{cl411, cl426, cl422, cl418, cl410\} < ROOMS$

Intention : This domain contains all the rooms that have to be considered.

Domain $HALLWAYS = \{h1, h2, h3\} < HallwaySection$

Intention : This domain contains all the hallway sections that have to be considered.

Requirements/Specifications

Property P_F_1

Formal : $\Box(\forall r \in ROOMS : (\forall s \in ROOMS :$
$r.RT.cp.tCPFM.heatingPeriod =$
$s.RT.cp.tCPFM.heatingPeriod))$

NL : The heating period is the same for all rooms.

Property P_F_2

NonFormal : Energy consumption should be reduced as far as possible during maintaining of comfortable working conditions for all users.

Property P_F_3

NonFormal : The control system shall be designed such that it can be easily changed and expanded.

Index

P

R

S

T

U

V

W