A STATECHART TEMPLATE LIBRARY FOR IOT SYSTEM MODELLING

Clyde Rempillo
Sadaf Mustafiz

Department of Computer Science
Toronto Metropolitan University
245 Church Street
Toronto, ON, CANADA
{clyderemp,sadaf.mustafiz}@torontomu.ca

ABSTRACT

In this paper, we propose a library of statechart templates for designing IoT systems. We have developed atomic statechart components modelling the heterogeneous aspects of IoT systems including sensors, actuators, network, and controller. Base system units for smart systems have also been designed. A component for calculating power usage is available in the library. Additionally, a smart hub template that controls interactions among multiple IoT systems and manages power consumption has also been proposed. The templates aim to facilitate the modelling and simulation of IoT systems. Our work is demonstrated with a smart home system consisting of a smart light, a smart microwave, a smart TV, and a smart fire alarm system. We have created a multi statechart with Yakindu based on the proposed templates and components. A smart home simulator has been developed by generating controller code from the statechart and integrating it with a user interface.

Keywords: Model-based design, Statecharts, Templates, Internet-of-Things (IoT), Smart home system.

1 INTRODUCTION

The Internet of Things (IoT) paradigm has brought about new challenges in software and systems engineering due to the complex, heterogeneous systems that are required to cooperate reliably in an uncertain and ever-changing environment. IoT is revolutionizing application domains from consumer and commercial to industrial and infrastructure (Atzori, Iera, and Morabito 2010). Smart home systems have gained immense popularity with applications such as, smart lights, smart appliances (smart fridge, smart washing machine, smart microwave, etc.), smart TV, and smart garage. Smart home automation aims to control the numerous systems used within a home as well as monitor energy consumption autonomously (Gunge and Yalagi 2016).

The different facets of IoT - software, hardware, network, and environment - need to be addressed in the design of such complex systems. The cyber-physical and heterogeneous nature of things and the wide array of possible applications make simulation essential for the design and deployment of smart services (D'Angelo, Ferretti, and Ghini 2017, Kecskemeti et al. 2017). We also need to design our system using the most appropriate modelling languages at the most appropriate levels of abstraction.

Statecharts (Harel 1987) are commonly used for behavioural modelling, simulation, and code synthesis. The behaviour of each smart system along with the coordination of the different systems can be modelled and simulated as a multi state machine system. The complex design of such systems includes various generic components modelling the hardware, software, network, and environment, and the interplay among them. Having templates that provide guidelines and gives a head start in modelling IoT/CPS system behaviour would be quite beneficial for designers. This would also be helpful for modellers who do not have deep knowledge on the heterogeneous components of such systems.

In this paper, we propose a statechart template library to model the behaviour of IoT systems. The templates come with a library of atomic components to model various aspects of IoT systems, namely sensors, actuators, controller, network, as well as an additional component to be used for modelling power consumption. The goal of the template is to facilitate the process of modelling complex IoT systems by assisting designers with building multi state machines to detail the system behaviour and to reduce development time and effort. As an illustrative example, we developed a training simulator for a smart home system using our library of templates.

This paper is structured as follows: Section 2 provides essential background information. Section 3 outlines the running example of the smart home application. Section 4 presents the statechart templates for IoT. Section 5 demonstrates the application of our templates to the smart home system. Finally, Section 6 discusses related work and Section 7 concludes the paper and mentions some future work.

2 BACKGROUND

Our work in this paper is based on the Statecharts language. In this section, we provide a brief background on Statecharts and our target application domain, IoT.

Statecharts. The Statecharts formalism is an extension of Deterministic Finite State Automata with hierarchy, orthogonality and broadcast communication (Harel 1987). It is a popular formalism for the modelling of the behaviour of reactive systems. It has an intuitive yet rigourously defined visual notation and semantics. It is the basis for documentation, analysis, simulation, and code synthesis. A statechart model is usually described with the following basic elements: states (basic, orthogonal, composite), transitions (event-based or time-based), enter/exit actions, history state, guards or conditions, and actions.

YAKINDU (Itemis 2022) (recently renamed to itemis CREATE) is a modular toolkit for modelling and simulating statecharts. It also supports generation of executable finite state machines (FSM). YAKINDU Statechart Tools (YAKINDU SCT) is based on Harel's Statecharts and supports the Statemate semantics introduced by Harel. It also supports multi state machine modelling that enables complex behavioural models to be modularized by splitting the model into smaller state models. This allows for separation of concerns and for the models to be reused by embedding in other statecharts. Multiple instances of a given statechart can be created. A system can thus be modelled as multiple collaborating statecharts. YAKINDU statecharts hold structural properties in addition to the behavioural aspect. These properties include variables and events that define the interface of the statechart. YAKINDU provides code generation support for the Java, Python and C languages among others.

IoT/Cyber-Physical Systems (CPS). In this paper, we follow the unified definition of IoT/CPS proposed in (Greer et al. 2019): *Internet of Things and cyber-physical systems comprise interacting logical, physical, transducing, and human components engineered for function through integrated logic and physics.*

The *things* constituting an IoT system (or corresponding "physical" entities in a CPS) have common goals and need to interact and cooperate to fulfill the functionalities of a smart system. The need to communicate with and control physical devices assigns new characteristics to such systems. The many facets of IoT - software, hardware, network, and environment - have to be taken into consideration during system design.

An IoT/CPS system has a logical state and a physical state. A change in physical state trigger sensors to produce values for the logical system, which may lead to changes in the logical state. This may in turn trigger actuation events that lead to a change in physical state.

In this paper, although we refer to the target domain as IoT, our work applies to both IoT and smart CPS.

3 RUNNING EXAMPLE: SMART HOME SYSTEM

In this section, we present our running example of a smart home application, which is a representative case for IoT systems (Asadullah and Raza 2016). Our smart home system is composed of a smart fire alarm system, a smart lights system, a smart TV system, and a smart microwave system. The latter two are not included here due to space constraints.

3.1 Smart Fire Alarm System

The *Smart Fire Alarm System* continuously monitors the heat, smoke, and carbon monoxide levels within an environment. We simulate a home environment which is monitored by the *Smart Fire Alarm System*, and if any risk of danger is sensed by the sensors, the fire alarm is activated. The following describes the behaviour of the system:
- Smart Fire Alarm System must always start from a *safe* state.
- Each *sensor* component is continuously monitoring the environment for either smoke, heat, or carbon levels.
- By sensing the smoke, heat, and carbon levels, the sensors will output the readings and sending data to the controller. When there is a change in the readings (for instance, due to any danger presence), the controller checks if the threshold value is crossed.
- If the threshold is reached, then the system transitions to a *warning* state and activates the *timer*. The Smart Fire System sends this warning alarm to the Smart Hub System to allow the *user* to check the danger presence and act accordingly. The system provides two warning states: *initial warning* and *final warning* with corresponding timers. On timeout, the system moves from initial to final warning, and then after a given time the fire alarm is activated.

3.2 Smart Lights System

The *Smart Light System* is an IoT system composed of sensors and actuators. The *light system* also has an integrated *WiFi* component that simulates an internet gateway connection to the server, which allows the hub to connect to the system. An additional *Power* component is also implemented as it helps calculate the *kWh* consumption of the system. The following describes the behaviour of the system:
- Smart Lights System will always start from an off state, and can be turned on manually or via the *sensors* integrated within the system.
- Using the *sensor* component, it will detect physical movements, then the *controller* component will handle this and trigger an actuating event in order to activate the light.
- While the lights are on, it will have their own individual timeout value caused by inactivity which is continuously monitored by the sensors. After this times out, the lights will automatically be turned off.

3.3 Smart Hub System

The Smart Hub System is an IoT system that acts like a hub and integrates multiple smart systems and allows each one to communicate and be controlled through the hub. In our running example, the smart hub is composed of these smart systems: Smart Fire Alarm, Smart Lights, Smart TV, and Smart Microwave. Each system works autonomously and are interconnected through the Smart Hub, which acts as the central terminal that manages coordination of the systems. The Smart Hub device enables the user to switch on/off the smart systems, with the exception of the smart fire alarm, remotely.

If the Smart Fire Alarm System activates the fire alarm when fire is detected, then it will notify and alert the hub. Then, the Smart Hub will process this emergency procedure and deactivate all of the other connected systems and restrict them from being operated until the fire alarm is turned off and safety is confirmed.

Other hub features such as *Power Manager* is also implemented to manage the power consumption level of the home environment and ensure that the total power consumption level stays under threshold. If the total consumption crosses the threshold, the manager recognizes this and turns off the most consuming system until the total is under threshold again.

4 STATECHART TEMPLATES

In this section, we propose a library of templates and components for modelling the behaviour of IoT systems. The hub and IoT templates are skeletons used for building statecharts for smart systems for the purpose of simulation. They can be easily modified or extended at any time according to the complexity of the system.

The library can be utilized to model various smart IoT systems without having to build from scratch. This library contains a growing collection of components and base systems that can be considered as building blocks towards developing complete, sophisticated, and complex IoT systems.

A component can be of two types: 1) atomic, and 2) base system unit. Using the IoT Template, we follow the structure within the template and import specific components to essentially build the system under study. For example, for the Smart Fire System, we used the IoT Template to design the full system, however, we imported the atomic components, namely the sensor, actuator and controller. We also used the base system unit for the Fire Alarm System. Essentially, without the integration of the IoT template and the components, the base units are just individual units working autonomously. Moreover, we also propose a Smart Hub Template for IoT systems that play the role of a coordinator or hub for multiple systems. The hub template can be extended by adding desired systems.

The library of templates has been implemented with YAKINDU Statechart Tools. YAKINDU enables the smart system models built with our library to be simulated, and statechart code to be generated from the models. A video demonstrating the template library is available at https://mde-tmu.github.io/iot-sc-template-library/#demo-videos.

4.1 IoT Template

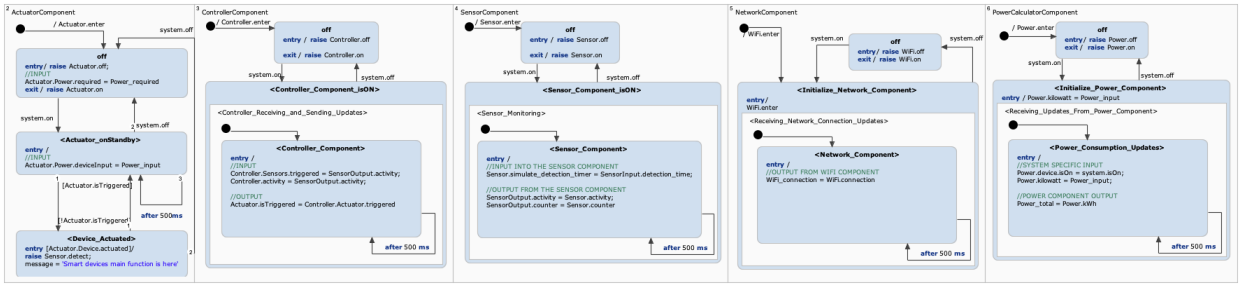


Figure 1: IoT statechart template (Extract).

The IoT Statechart Template (shown in Fig. 1) contains orthogonal regions that are used to import the atomic components from the template library along with the pre-designed base system units into their designated regions. Essentially, the IoT Template allows users to design IoT systems using the pre-built templates and components. We have modelled the default template to have seven orthogonal regions, however, these can be extended depending on the system complexity. Once each system is modelled, it can then be imported

into the Smart Hub Template to design a Smart Hub System, which is used to interconnect and manage other smart systems. All systems enter a default state that contains the seven orthogonal regions. These regions have composite states that contain the components with the initial state `off`. When the event *system.on* occurs, this will also raise the *on* events in all connected systems.

The `DeviceSwitchStatus`, the states `on` and `off` of the system is modelled (not shown in Fig. 1). These transitions are triggered via the interface. The template has dedicated regions for the `Sensor`, `Controller` and `Actuator` components. We begin with the `Sensor` component monitoring by raising a timed transition event periodically (set to every 500 millisecond/ms). The `Sensor` sends a signal (i.e., a boolean activity value) to the `Controller` every 500 ms. Once the `Controller` receives a signal indicating activity, it will execute a decision-making process by coordinating with the *base system unit* and output a signal to raises an actuating event. This enables a transition in the `Actuator`. For example, the Smart Light System will actuate the lights to turn on.

4.2 Smart Hub Template

The Smart Hub Template (shown in Fig. 2) is composed of pre-defined orthogonal states and regions that are used to manage the coordination between multiple smart systems. Figure 2 shows the regions for each IoT system controlled by the hub. To simulate a network connection between the hub and the systems, we have also added the `Network` component in the hub.

When applying the template, more systems can be added by adding an orthogonal region per system and reusing the state machine defined in the template. The template covers some basic functionalities: 1) turning on and off individual smart systems (see first region from the left in Fig. 2), 2) turning on and off all of the smart systems at once, 3) calculating the total power consumption using the `Hub Power Manager` component, (see second region from the left in Fig. 2), and monitoring all smart system status through the hub.

When the hub is turned on via an external event, the statechart enters multiple orthogonal regions in the `HubTemplate` state (see Fig. 2). The right-most orthogonal regions in the template are assigned to the IoT systems that are imported into the hub. These regions enter initial states and triggers an action to invoke the corresponding multi state machine with the *statechart.enter* event. The left-most region contains a composite state that contains substates indicating the status of the connected systems. Each system enters a default state *off* and can be turned on/off either individually or simultaneously by raising the event *HUBAllSystemsON*/*HUBAllSystemsOFF*.

4.3 Atomic Components

The IoT statechart template includes a composition of components that defines an IoT smart device or system. In order to properly design an IoT system, we must model the constituent atomic components. These are designed to exhibit the behaviour of the heterogeneous software, hardware, and network components of an IoT system. We have modelled the IoT Template to be composed of the following atomic components: `device, network, sensor, actuator, controller`, and `power`. The statechart for each atomic component in the library, presented in Fig. 3, has been developed and included in our IoT library. `Sensor` models the behaviour of sensors. A `sensor` component is periodically "sensing" or monitoring an environment or device. As shown in Fig. 3a, the `sensor` component enters a default composite state `Sensor Monitoring` when the system is on. When the system turns off, the event `toggle` is raised leading to a transition to the `off` state. When in the `sensor monitoring` state, a *true* value is assigned to the *reading* variable, which indicates that the `sensor` is now reading/monitoring. The initial state, `NoActivitySensed`, indicates that there is no activity being sensed. For simulation purposes, we have set a counter value that is used to enable a *timed transition* to the `ActivitySensed` state. The `Sensor` will then send this data to the `Controller` component.

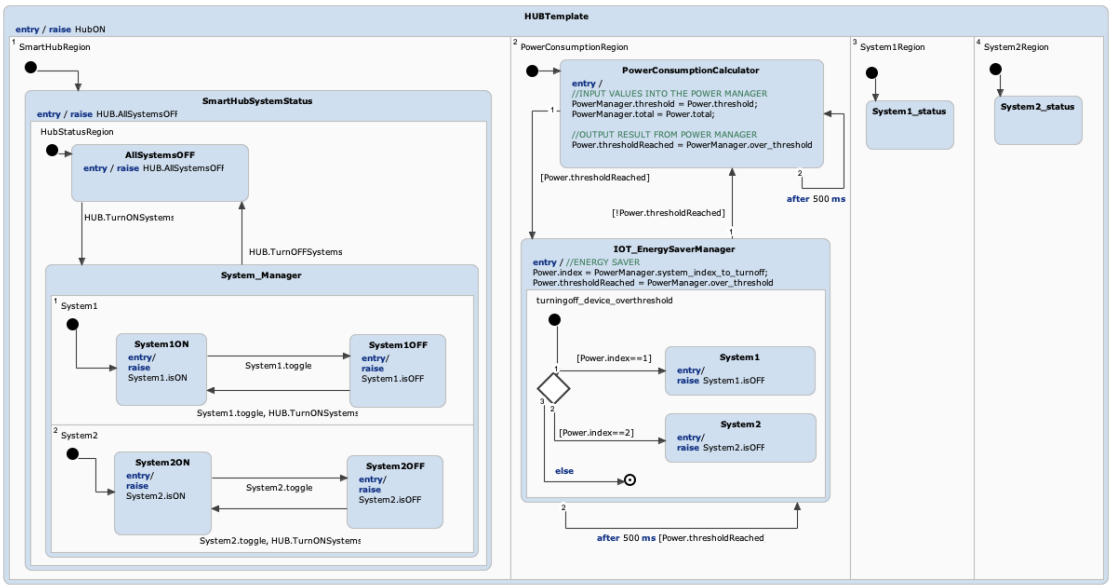


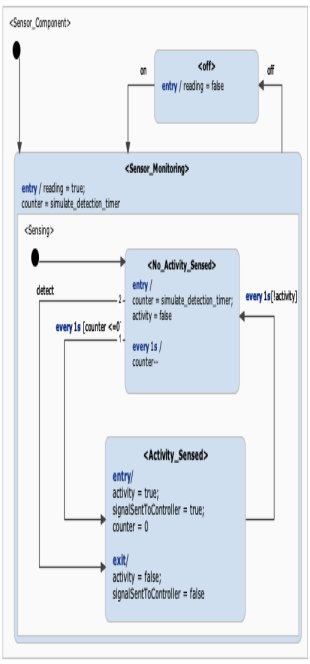
Figure 2: Smart hub template statechart.

`Controller` controls the behaviour of a system by receiving data from the sensors, making decisions, and raising events to trigger the `Actuators`. The `controller` will stay in the default entry state `WaitingforSensorData`, and can only transition when the conditional event `SensorsTriggered` is raised. When the `controller` receives the data from the `Sensor`, the `Controller` processes this to perform some decision making while entering the `SensorDataReceived` state. Once the `Controller` has processed the data, it will raise an actuating event, `ActuatorTrigger`, in the `Actuator` component. Once the `Controller` has communicated with the `Actuator`, it will transition to the *TriggerActuator* state until the conditional event is raised, and will then transition back to the default state *WaitingForSensorData*.

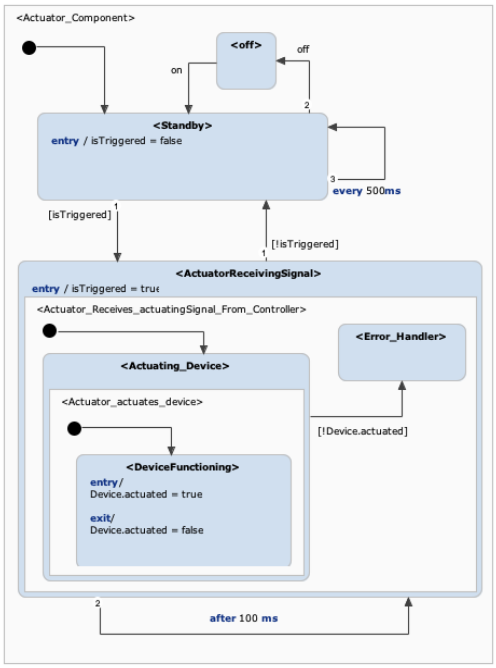
`Actuator` handles the activating/actuation of devices within a system. It receives an actuation signal from the `controller` and uses this to handle the activation of the system. As shown in Fig. 3b, the `Actuator` starts in the initial state, `StandBy`, as it waits for an *ActuatorTriggered* event from the `Controller`. Once the `Controller` sends a *true* value, then the `Actuator` will transition to the `ActuatingDevice` state.

`Network` handles the gateway connection between the system and the network. Each system may have a different network type such as *cellular, Wi-Fi, LPWAN, bluetooth* and *Zigbee.* Currently, only the statechart for Wi-Fi is included in the library. The `Network` component starts in an initial state `NetworkComponentWorking`, and can transition to the `off` state by raising the `off` event. Otherwise, it transitions and enters a composite state `checkingForNetworkConnection` and the default substate `connectingToServer`, which will indicate either a *successful* or *failed* connection using a conditional event. Additionally, for simulation, we have modelled the component to have a periodic network timeout.

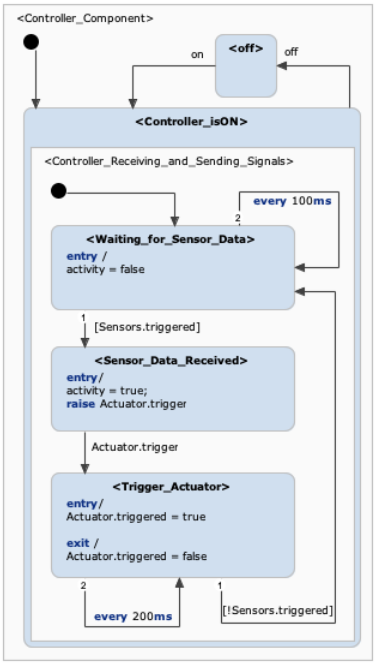
`Power` is an atomic component that handles the per-hour power consumption (in KWh) of the system. It calculates the total consumption of the system and this data is sent to the `Hub Power Manager` through the `Smart Hub System`. It contains a composite state the describes the consumption status of the system. The statechart has a default entry state *NoPowerConsumed* as each system always starts in the *off* state. Once the system is turned on, this will also reflect in the `power` component and raises the conditional event. The substate will then transition to the `ConsumingPower` state as it calculates the per hour consumption using the kilowattsPerHour(kWh) units.



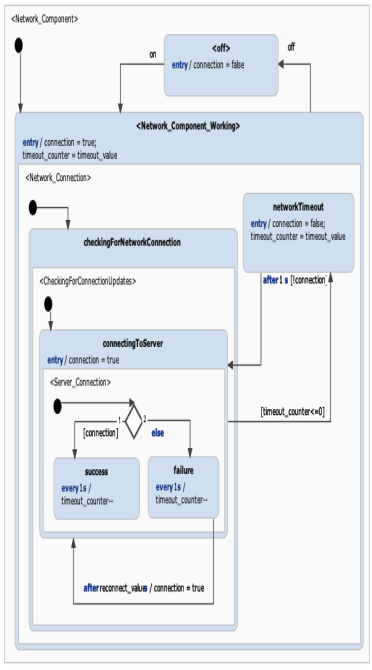
(a) AC: Sensor Component



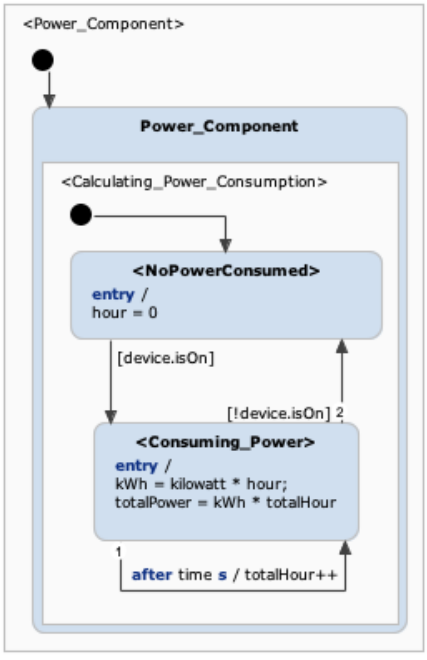
(b) AC: Actuator Component



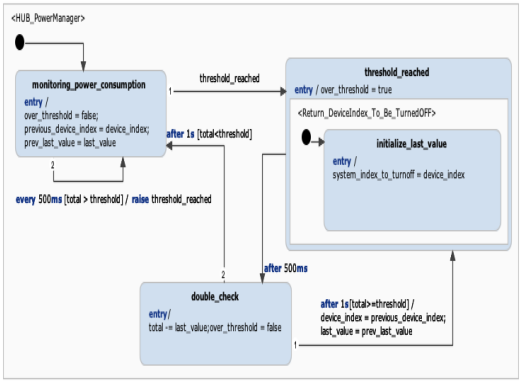
(c) AC: Controller Component



(d) AC: Network Component



(e) AC: Power Component



(f) MC: Hub Power Manager

Figure 3: Atomic Components (AC) and Manager Component (MC).

4.4 Base System Units

The **Base System Units** library is a collection of pre-designed units that can function autonomously without any smart features. These units (not shown here due to space constraints) can be imported from the library and are to be treated as self-contained systems that can function on their own but does not contain any specific “smart" functionalities. These smart functionalities are to be added using the atomic components from the IoT library. The goal is to build a library of reusable units over time.

4.5 Manager Components

The **Manager Components** are pre-designed components that contain special manager features that can *only* be imported into the Smart Hub Template. An example of a managing component is the `Power Hub Manager`. The `Power` component designed within the IoT systems calculates the system's power consumption and sends data to the hub. Whenever a device changes its status (e.g., from *on* to *off*), the total power consumption is updated immediately. Once the power threshold is reached, the Hub Power Manager transitions to the state `CheckingDevicePowerContribution` to check which device contributes the most to the total power consumption. Once the device consuming maximum power is identified, the hub will turn it off. This can be observed in Fig. 3f.

5 MODELLING THE SMART HOME SYSTEM CONTROLLER

In this section, the proposed library of templates are used to model the smart home system, introduced in Sec. 3. Using the IoT Template, we have designed multiple smart systems and connected all the systems to a smart home hub by designing a main controlling hub using the `Hub Template`. For space reasons, the smart microwave and TV models are not included in the paper.

5.1 Using the Templates

To use these templates, the user must be able to understand the structure and design of the templates. In the IoT template, we have designed it to contain all of the necessary atomic components (ie. `sensor, controller, etc.`). Every region and composite state are pre-configured with the variables, events, and transitions, hence, no changes are required in the atomic components. The following are guidelines for modellers using our library of templates.

1. Modeller must refer to the *IoT Template* when designing a smart system. If the modeller desires to model a hub to link smart systems together, then the *Hub Template* must be used.
2. Once a template is chosen, the modeller must determine which states and regions are editable.
3. For the *IoT Template*, the modeller can only edit the model by adding more `atomic components` and the `base system units`. The user may refer to the `atomic components` library and import any additional components.
4. The application-specific functionality has to be added as a *base system unit*.The modeller can refer to the *base system unit* library for this, however, if the desired unit is not available, then the user will have to design their own. Adding a unit means importing the *base system* statechart via the configuration panel, and then assigning an appropriate variable to the statechart. This will allow for an easier use of the *base system*'s built-in variables and events. Once the *base system* and the *components* are finalized, then the statechart can now be used as a smart system.
5. For the *Hub Template*, the modeller only needs to edit the number of smart systems connected. The hub can be connected to numerous smart systems as long as it is imported and extended properly. This can be done by importing the smart system statecharts via the configuration panel and assigning these to specific system variables. Doing this will allow accessibility to the smart system variables and events from the Smart Hub level.

5.2 Smart Home Statecharts

In this section, we present the statechart models developed for the smart home application with the use of our template library.

5.2.1 Smart Home Hub Statechart

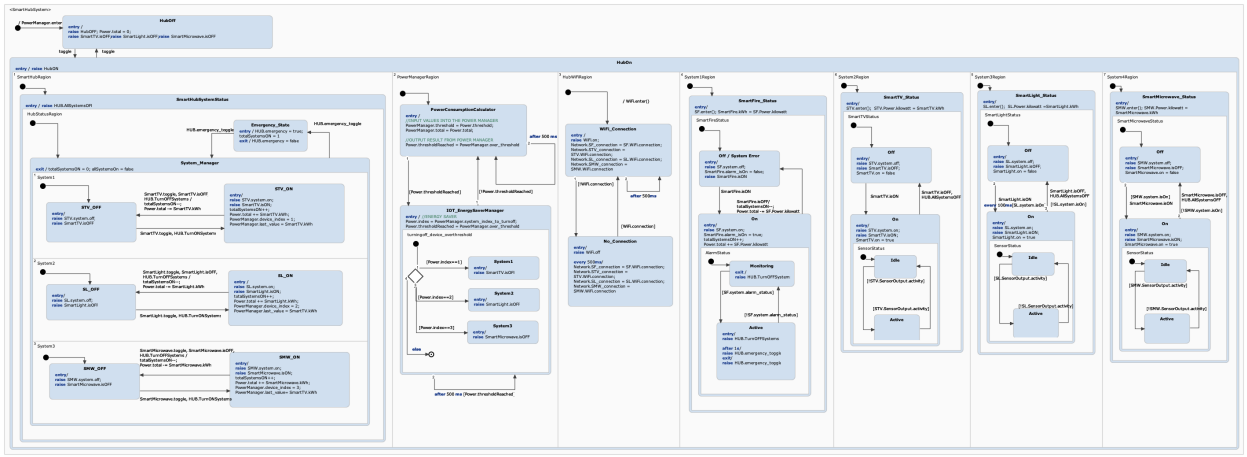


Figure 4: Smart home hub statechart model.

The *Smart Home Hub* statechart, designed using the *Hub Template*, aims to connect and link all smart systems located within a smart home system as shown in the orthogonal regions in Fig. 4. Once the hub system enters the *on* state, all of the orthogonal regions will also enter all of the corresponding initial states. For example, in the left-most region of Fig. 4, the initial state indicates that all systems, excluding the smart fire system, will enter the `off` state. The hub system has a *PowerManager* component, which basically monitors the total power consumption of the system (see second left region of Fig. 4. The power manager continuously monitors the total power and only exits from the `PowerConsumptionCalculator` state when the event `PowerThresholdReached` is raised in the `HubPowerManager`. Once this event occurs, the state transitions to *IoTEnergySaverManager*. In this state, it receives the system index output by the `HubPowerManager` and turns off the system that matches the index value.

5.2.2 Smart Fire Alarm Statechart

The Smart Fire Alarm System is composed of the main atomic components, namely `sensor`, `actuator`, and `controller` (see Fig. 5). We have specifically added three sensors for carbon, smoke and heat. We use the `IoT Template` as the main skeleton for system design.

The basic functionalities of a smart fire system are pre-designed within the base *Smart Fire* unit and is imported into the template (see Fig. 6). Once imported, the predefined atomic components will work accordingly. For simulation purposes, random detection times are assigned to the sensors (via the UI). Once the detection occurs, this raises an event that enables a transition to the *ActivitySensed* state within the `Sensor` component. The `Controller` will then process the input from the sensor in the *base system unit* statechart and monitor the sensor levels by raising the *SensorTriggerSignalReceived* event and transitioning to the *SensorTriggered* state. The system will also enter a `warning state` by raising the *Warning* event until the threshold value for the `sensor` is reached. Once the threshold is reached, the `controller` will send an actuation signal to the `actuator` to sound the alarm by raising the *Danger* event, which sends the *base system* to `Danger` state. When the `actuator` receives the actuation event, the fire alarm is triggered. Furthermore, the system will automatically send an emergency signal event to the `Smart Hub System`, which would send the hub to the `Emergency` state. Consequently, the `Smart Hub` will handle this emergency event and restrict all other systems within the environment from being used in the

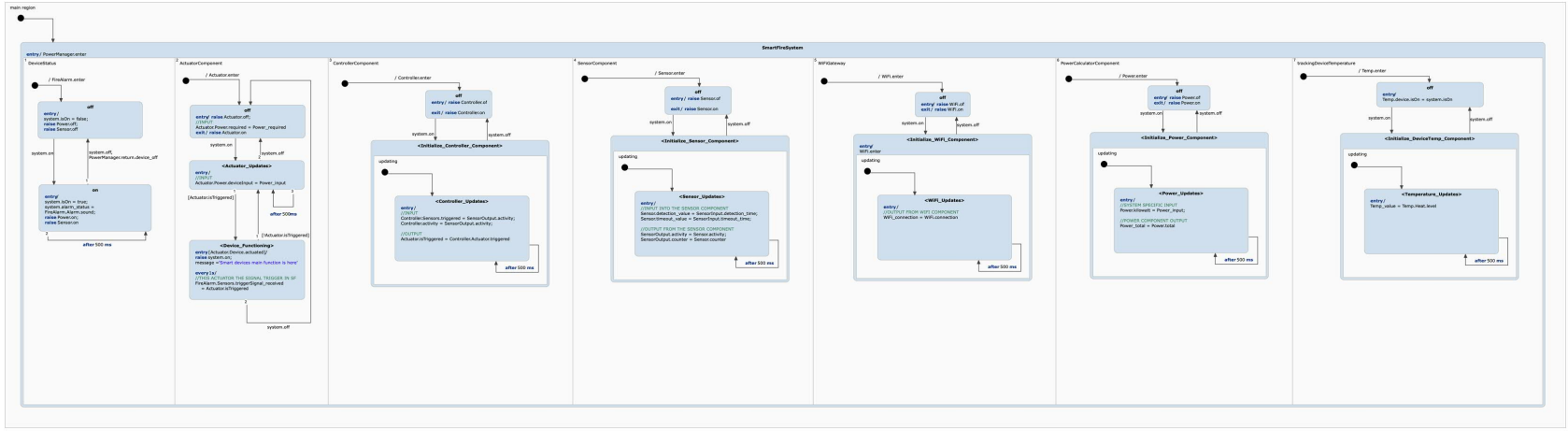


Figure 5: Smart fire alarm statechart model.

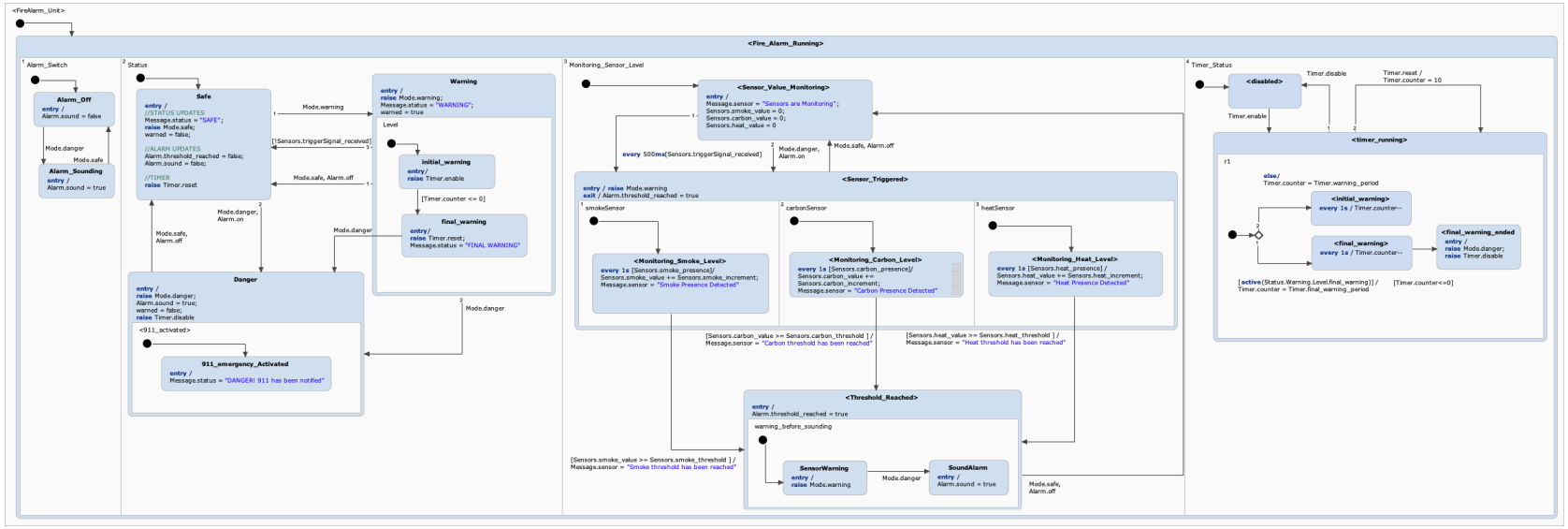


Figure 6: Base fire alarm unit.

`EmergencyState`. This can only be turned off once the fire alarm is turned off through the Smart Fire System.

5.2.3 Smart Lights Statechart

The Smart Lights System is also implemented with the atomic IoT components, `sensor`, `actuator`, and `controller` from the library. The basic light functionalities are defined within the `Base LED Lights unit` and is imported into the template. The base unit has the main functionalities such as `on/off`, `dim/brighten lights`. The Smart Light System's `sensor` monitors the environment for activity. Once activity is sensed, the `Sensor`'s activity data is received by the `Controller` component. The `Controller` in the base system unit recognizes and processes this data, and then raises an actuating event to turn the lights on. Additinally, once the `Sensor` outputs a false value, the `Controller` will raise another actuating event and transition back to the `standby` state.

5.3 Code Synthesis and Integration with the User Interface

For our training simulator, we have developed a dashboard with Java that serves as the user interface (UI) for the smart home hub. Using the code generation feature of Yakindu, we generated Java code from the statecharts. The statechart code is then integrated with the graphical Java-UI of the smart home. A video demonstrating the smart home system developed with our templates is available at https://mde-tmu.github.io/iot-sc-template-library/#demo-videos.

6 RELATED WORK

There exists a wide array of work on smart home automation focusing on IoT components, hardware platform, and implementation aspects or on the use of artificial intelligence (Williams, Terence J., and Immaculate 2019, Gunge and Yalagi 2016). These work have helped us understand the architecture of IoT systems and derive the core components to integrate in our templates. In this section, we discuss existing research work and projects on IoT and CPS modelling and simulation.

SysML4IoT (Costa et al. 2016) is a design and analysis approach supporting model checking of finite state machine models of IoT applications. ThingML (Harrand et al. 2016) is an MDE method for IoT that supports platform-independent modelling with a textual variant of statecharts that is used as the basis for generating controller code. Almeida et al. 2017 propose an approach for developing multimodal multi-device applications using SCXML state machines for controlling interaction flow. While these methods employ statechart modelling for specific applications, they do not propose any templates for IoT systems.

Rouillard and Tarby 2011 also use statecharts for modelling the interactons in a smart home, however the model is not used for simulation or code synthesis. Sinha et al. 2017 propose parametric statecharts for platform-independent modelling to support flexibility in designing mobile-Health applications. These models are then transformed to static statecharts which are then used for code synthesis for various mobile platforms. Hussein et al. 2017 make use of SysML4IoT to specify system functions and state machines to identify the adaptations to a given environmental context in IoT systems. They designed and deployed a smart lights system using their approach. All of these work design the statecharts from scratch. Using our library that caters to various smart systems, designers would get access to reusable SC components that they can build on. In the modelling and simulation area, Matlab Simulink and Stateflow are popularly used for IoT/CPS. Maryasin et al. (Maryasin et al. 2019) propose a mathematical modelling and simulation approach for smart buildings using Simulink and Stateflow. Simscape (Matlab 2012a) library is used for creating block-based models composed of physical components, such as tanks, motors, pumps, etc. Stateflow library (Matlab 2012b) offers design patterns for supervisory logic and application, e.g., schedulers and fault detection.

In the context of smart hub systems, Béguery et al. 2011 provides applications of hub-centered system that link all smart system within a smart building. They have designed their simulation models with Matlab SimuLink specifically for their target application. In our work, the statechart models are templates that can be used to design many different hub-centered systems.

While statecharts have been used extensively for modelling and/or for simulation of CPS and IoT systems, however, to the best of our knowledge, there are no existing works on reusable statechart templates for modelling and simulating the behaviour of IoT systems.

Our proposed library aims to provide a coherent set of extensible and maintainable domain-specific statechart models that can reduce complexity and facilitate the modelling and simulation tasks for IoT systems and software engineers.

7 CONCLUSION

The paper proposes a library of statechart templates to model the behaviour of cyber-physical and IoT systems. Two templates, *IoT template* and *Hub template*, have been developed that provide skeletons for modelling statecharts for smart systems as well as smart hubs (e.g, a smart hub coordinating multiple smart systems in a home). An extensible library of basic atomic components modelling the different aspects of IoT systems, including hardware, software, and network, has been proposed. A generic statechart template that is reusable for various IoT systems has been designed for each of the atomic component, namely, sensor, actuator, controller, network, and power calculator. We demonstrate the use of our templates with a representative case of a smart home system composed of multiple smart systems (smart lights, smart TV, smart microwave, and smart fire alarm). The template can be used to model systems outside the smart home domain.

As a next step, we intend to extend the set of templates to cover systems with a broader range of components, hardware devices (e.g., tags and physical entities) and network connectivity (e.g., LPWAN and Zigbee). For further evaluations, we plan on using the templates to design training simulators for more complex systems, such as a smart city. Moreover as future work, we will be modelling the architectural view of the system along with the statechart model to go towards synthesizing code for deployment based on the architecture.

ACKNOWLEDGMENTS

This work is partly funded by NSERC. The authors would like to thank Prof. Hans Vangheluwe for his valuable insights on this work.

REFERENCES

Almeida, N., S. Silva, A. Teixeira, and D. Vieira. 2017. “Multi-Device Applications Using the Multimodal Architecture”. In *Multimodal Interaction with W3C Standards: Toward Natural User Interfaces to Everything*, edited by D. A. Dahl, pp. 367–383. Cham, Springer International Publishing.

Asadullah, M., and A. Raza. 2016. “An overview of home automation systems”. In *2016 2nd International Conference on Robotics and Artificial Intelligence (ICRAI)*, pp. 27–31.

Atzori, L., A. Iera, and G. Morabito. 2010. “The internet of things: A survey”. *Computer networks* vol. 54 (15), pp. 2787–2805.

Béguery, P., Y. Lamoudi, O. Cottet, O. Jung, N. Couillaud, and D. Destruel. 2011, 01. “Simulation of Smart Building - HOMES pilot site”.

Costa, B., P. F. Pires, F. C. Delicato, W. Li, and A. Y. Zomaya. 2016. "Design and analysis of IoT applications: a model-driven approach". In *IEEE 14th Intl Conf on Dependable, Autonomic and Secure Computing (DASC/PiCom/DataCom/CyberSciTech)*, pp. 392–399. Institute of Electrical and Electronics Engineers, Inc.

D'Angelo, G., S. Ferretti, and V. Ghini. 2017, July. "Modeling the Internet of Things: a simulation perspective". In *International Conference on High Performance Computing Simulation (HPCS)*, pp. 18–27.

Greer, C., M. Burns, D. Wollman, and E. Griffor. 2019. "Cyber-physical systems and internet of things". Technical Report Special Publication 1900-202, National Institute of Standards and Technology (NIST).

Gunge, V. S., and P. S. Yalagi. 2016. "Smart home automation: a literature review". *International Journal of Computer Applications* vol. 975 (8887-8891).

Harel, D. 1987. "Statecharts: A Visual Formalism for Complex Systems". *Sci. Comput. Program.* vol. 8 (3), pp. 231–274.

Harrand, N., F. Fleurey, B. Morin, and K. E. Husa. 2016. "ThingML: a language and code generation framework for heterogeneous targets". In *19th International Conference on MODELS*, pp. 125–135.

Hussein, M., S. Li, and A. Radermacher. 2017. "Model-driven Development of Adaptive IoT Systems". In *ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS 2017) Satellite Event*, pp. 17–23.

Itemis 2022. "Yakindu". https://www.itemis.com/en/products/itemis-create/. Accessed: September 15, 2022.

Kecskemeti, G., G. Casale, D. N. Jha, J. Lyon, and R. Ranjan. 2017. "Modelling and Simulation Challenges in Internet of Things". *IEEE Cloud Computing* vol. 4 (1), pp. 62–69.

Maryasin, O. Y., A. S. Kolodkina, and A. A. Ogarkov. 2019. "Computer Simulation of a Smart Building". *Automatic Control and Computer Sciences* vol. 53, pp. 787 – 793.

Matlab 2012a. "Simscape User's Guide". Technical report, The MathWorks Inc.

Matlab 2012b. "Stateflow User's Guide". Technical report, The MathWorks Inc.

Rouillard, J., and J.-C. Tarby. 2011. "How to communicate smartly with your house?". *Int. J. Ad Hoc Ubiquitous Comput.* vol. 7, pp. 155–162.

Sinha, R., A. Narula, and J. Grundy. 2017. "Parametric Statecharts: Designing Flexible IoT Apps: Deploying Android m-Health Apps in Dynamic Smart-Homes". In *ACSW*, ACM.

Williams, V., S. Terence J., and J. Immaculate. 2019. "Survey on Internet of Things based Smart Home". In *2019 International Conference on Intelligent Sustainable Systems (ICISS)*, pp. 460–464.

AUTHOR BIOGRAPHIES

CLYDE REMPILLO is an M.Sc. student in the Department of Computer Science at Toronto Metropolitan University. He finished his Bachelors in Computer Science from University of Windsor in 2021.He is currently working on modelling and simulation of smart systems. His email address is clyderemp@torontomu.ca.

SADAF MUSTAFIZ is an Assistant Professor at the Department of Computer Science at Toronto Metropolitan University. She received her Ph.D. and M.Sc in Computer Science from McGill University. Her research interests are primarily in the area of model-driven software engineering. Her email address is sadaf.mustafiz@torontomu.ca.