SAWAD:MAPE Design Patterns For Self-Adaptive WSN Applications

Research in Developing and Controlling Distributed Applications

1 Transformation Rules for Centralized MAPE design Pattern

The graphic transformation of the MAPE design pattern models seeks to automatically generate Event-B specifications. In this section, we define transformation rules to translate the different elements in the centralized MAPE design pattern to their respective concepts in Event-B notation. Structural features are specified with several contexts *Ci* and behavioral features are specified several machines *M*i.

1.1 Structural features

In what follows, we define the static part of the centralized pattern. It is specified in the Context part. The transformation of the centralized pattern is based on six major rules allowing the transformation of a graphical model into an Event-B specification.

Pattern is formalized in incremental way while using successive refinements to reduce the complexity of pattern formalization. The formalization process is composed of three refinement steps as shown in Figure 1. In the first step, an abstract pattern model is defined. In the next step, we add to the model the main MAPE components (M,A,P,E) and the different sub-components, then, we specify *Probe*, *Effector* and *ContextElement*. In a later refinement step, we add the connection through provided/required interface.

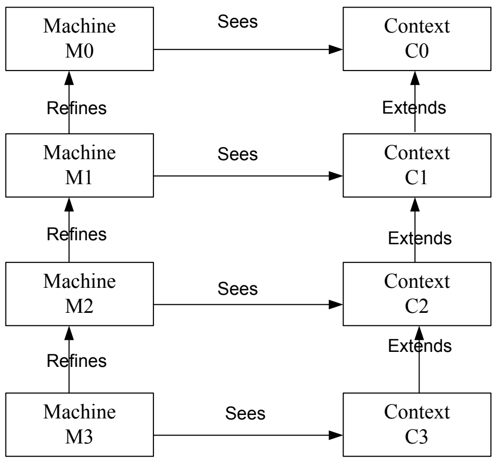


Fig. 1. Pattern refinement

- **Context C0:**

R1.Architecture nodes transformation rules: Participants that constitute the architecture of the centralized pattern are the Sink node and the Sensor node. Figure 2 shows how to transform the architecture entities.

R1.1. This rule transforms the sink node name *Sink* into constants in the *CONSTANTS* clause.

R1.2. This rule transforms sensor nodes name *Sensor$_i$* into constants in the *CONSTANTS* clause. The set of sensor nodes is composed of all sensors name. This is transformed formally to a partition (Sensors _partition) in the *AXIOMS* clause i.e. SensorsNodes = {sensor$_1$,...,sensor$_n$} ∧ sensor$_1$#sensor$_2$ ∧...∧ sensor$_{n-1}$#sensor$_n$

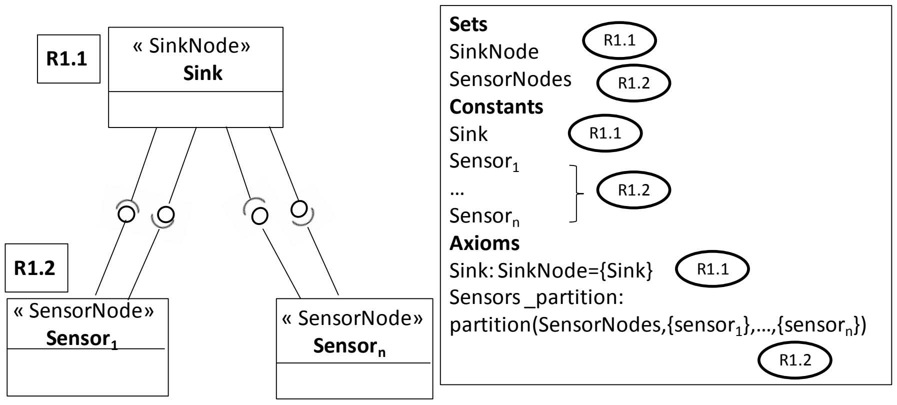


Fig. 2. Architecture nodes transformation rules

- **Context C1:**

R2. MAPE components transformation rules: The MAPE control loop is composed of four components: Monitor, Analyzer, Planner and Executor. The sink node centralizes Analyzer and Planner components .The sensor node contains the Monitor and Executor components.
Figure 3 shows how to transform MAPE components.

R2.1. This rule transforms Monitor components name M$_i$ into constants in the *CONSTANTS* clause. The set of monitor is composed of all monitor components name. This is transformed formally to a partition (Monitors_partition) i.e. MonitorsComponent = {M$_1$,...,M$_n$} ∧ M$_1$#M$_2$ ∧...∧ M$_{n-1}$#M$_n$.

R2.2. This rule transforms Executor components name E$_i$ into constants in the *CONSTANTS* clause. The set of executors is composed of all executor components name. This is transformed formally to a partition (Executors_partition) i.e. ExecutorComponents = {E$_1$,...,E$_n$} ∧ E$_1$#E$_2$ ∧...∧ E$_{n-1}$#E$_n$.

R2.3. This rule transforms components in the sink node (analyzer and planner) named ASink and PSink into constants in the *CONSTANTS* clause in the Event-B specification. This is transformed formally to a set (Sink_Set).

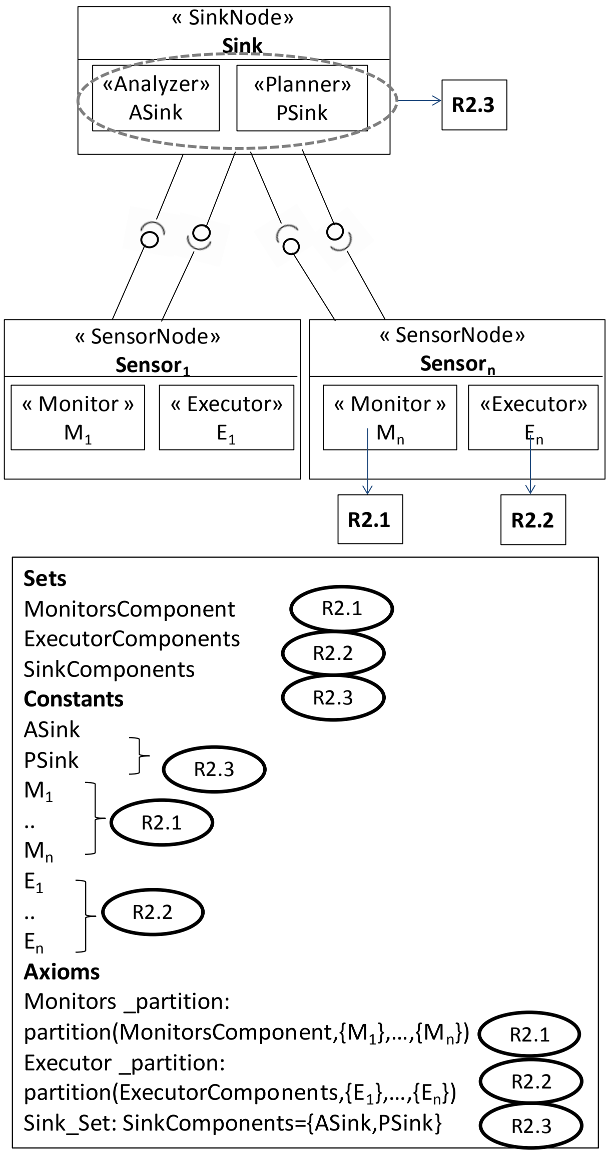


Fig. 3. MAPE component transformation rules

R3 MAPE Sub-component transformation rules*:* Each MAPE component is composed of three sub-components: a receiver, processor and sender. Figure 4 shows how to transform MAPE sub-components.

R3.1. The Monitor component is composed of:

- Event collector (EC): it collects events from sensors.
- Aggregation/filter functionality (AF): the role of this sub-component is filtering, i.e. discarding events that are deemed to be irrelevant for the managed application, and aggregating events.
- Symptom sender (SS): this sub-component generates a symptom that needs to be analyzed based on a list of events and lets them available to the Analyzer.

This rule (R3.1) transforms Monitor sub-component named: EC, AF and SS into constants in the *CONSTANTS* clause. The set of Monitor_Subcomponent is composed of all Monitor_Subcomponents name. This is transformed formally to a set (Monitor_Subcomponent_set) i.e. Monitor_Subcomponent = {EC,AF,SS}

R3.2. The Analyzer component contains:

- Symptom receiver (SR): it receives symptoms transmitted by the Monitor component.
- Symptom reasoner and analysis (SRA): it performs data analysis and reasoning on the symptoms provided by the Monitor component.
- RFC sender (RFCS): if degradation is detected and its source is identified, then a request for change is passed to the Planner component.

This rule transforms Analyzer sub-component named: SR, SRA and RFCS into constants in the *CONSTANTS* clause. The set of Analyzer_Subcomponent is composed of all Analyzer_Subcomponent name. This is transformed formally to a set (Analyzer_Subcomponent_set) where Analyzer_Subcomponent={SR,SRA,RFCS}

R3.3. The Planner component integrates:

- RFC receiver (RFCR): it receives published RFCs.
- Planner coordination (PC): based on policies, PC takes decisions that affect the managed application behavior and/or structure. It chooses the required actions to achieve goals and objectives.
- Plan sender (PS): it publishes generated plans to make them available to the Executor component.

This rule transforms Planner sub-components named: RFCR, PC and RFCS into constants in the *CONSTANTS* clause. The set of Planner_Subcomponent is composed of all Planner_Subcomponent name. This is transformed formally to a set (Planner_Subcomponent_set) where Planner_Subcomponent={RFCR,PC,PS}

R3.4. The Executor component includes:

- Plan receiver (PR): it receives logical description of the sequence of actions to be executed.
- Executor manager (EM): it translates to commands every step of the plan and guarantees the correct execution over the target system.
- Reconfiguration executor (RE): it sends the commands to execute over the target system.

This rule transforms Executor sub-components named: PR, EM and RE into constants in the *CONSTANTS* clause. The set of Executor_Subcomponent is composed of all Executor_Subcomponent name. This is transformed formally to a set (Executor_Subcomponent_set) where Executor_Subcomponent={PR,EM,RE}

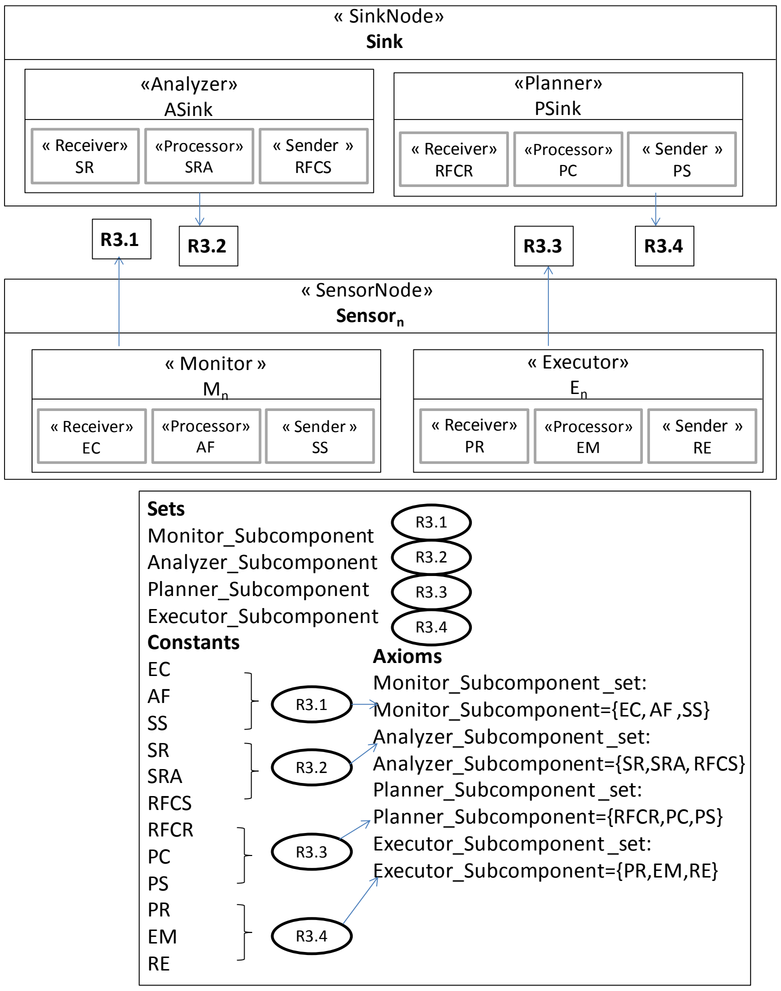


Fig. 4. MAPE sub-component transformation rules

- **Context C2:**

R4 Probe, ContextElement and Effector transformation rules:

R4.1. This rule transforms Probe name P_i into constants in the *CONSTANTS* clause in the Event-B specification. This is transformed formally to a partition (Probes-partition).

R4.2. This rule transforms ContextElement name Ctx_i into constants in the *CONSTANTS* clause in the Event-B specification. This is transformed formally to a partition (Context-partition).

R4.3. This rule transforms Effectors name E_{Si} into constants in the *CONSTANTS* clause in the Event-B specification. Also the set of Effectors is specified using a partition in the *AXIOMS* clause (Effector_partition), that is Effector = $\{E_{S1},...,E_{Sn}\} \wedge E_{S1}\#E_{S2} \wedge ... \wedge E_{Sn-1}\#E_{Sn}$.

Figure 5 shows how to transform *Probe*, *ContextElement* and *Effector components*.

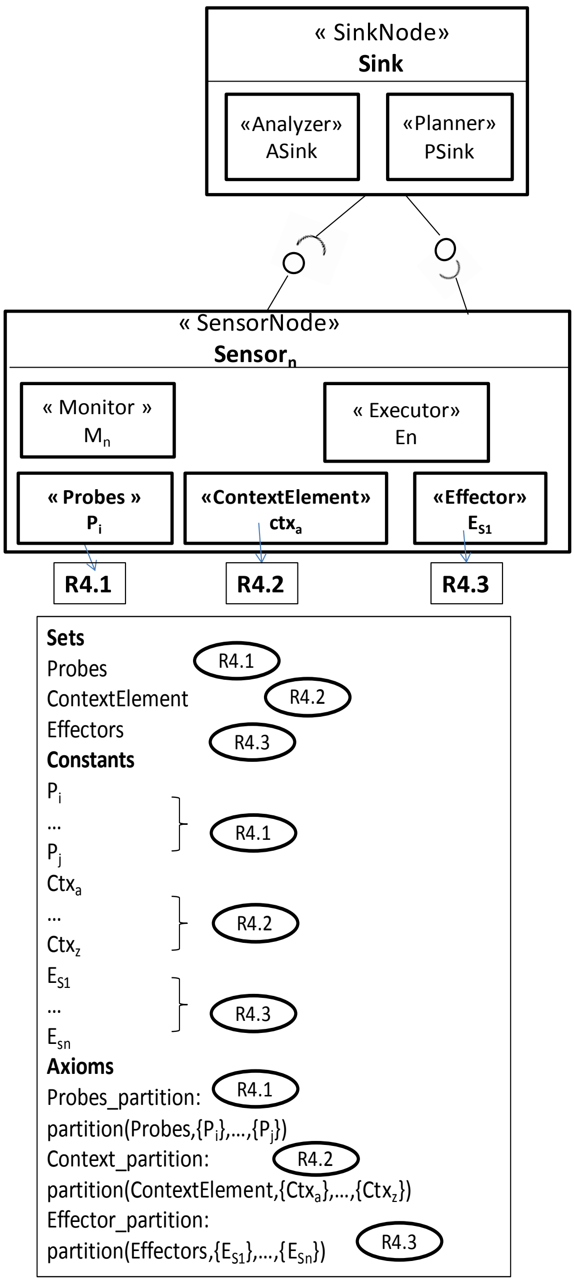


Fig. 5. Probe,ContextElement and Effector transformation rules

R5.Enumeration transformation rules: This rule transforms each enumeration name into set in the *AXIOMS* clause and enumeration literals are transformed into constants in the *CONSTANTS* clause. Figure 6 shows how to transform each Enumeration.

R5.1. The SymptomAnalysis enumeration is transformed into set in the *AXIOMS* clause in the Event-B specification. The enumeration contains Degradation and No Degradation enumeration literal. Degradation and No degradation are transformed into constants in the *CONSTANTS* clause in the Event-B specification. This is transformed formally to a set in the *AXIOMS* clause (SymptomAnalysis _set) i.e. SymptomAnalysis = {Degradation, No Degradation }

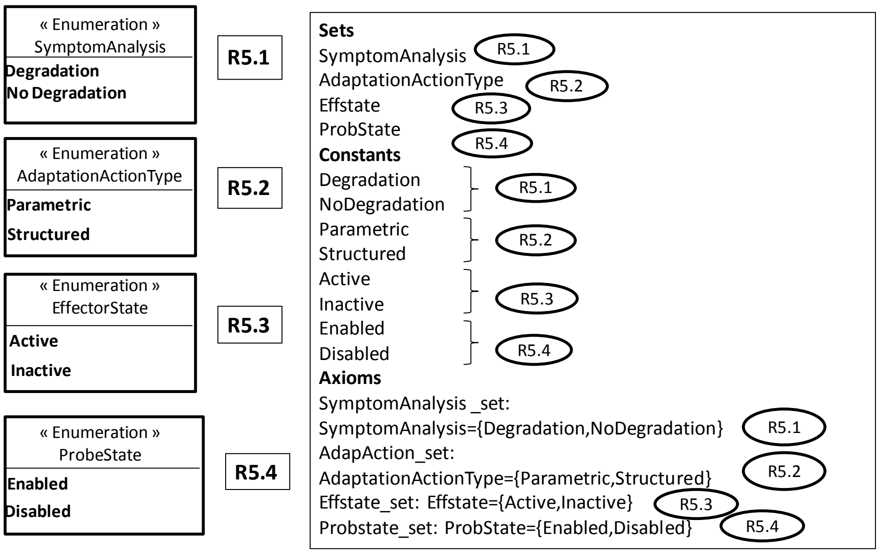
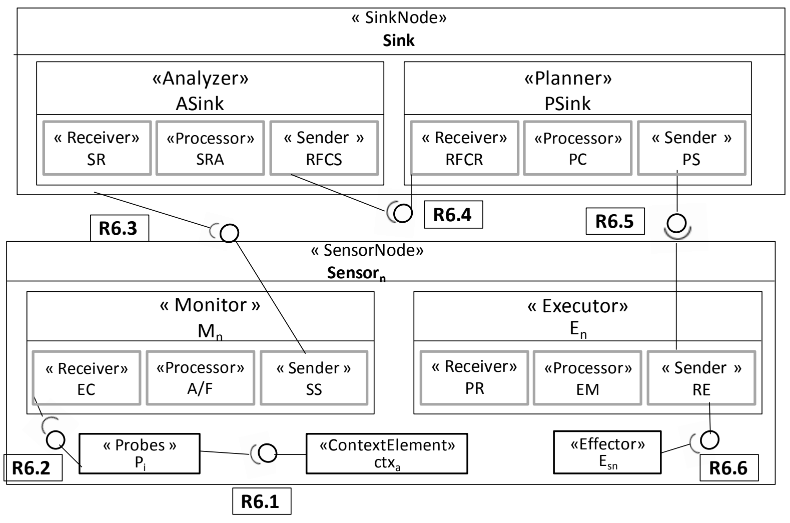


Fig. 6. Enumeration transformation rules

- **Context C3:**

R6. Interface transformation rules: This rule defines the graphical connection (provided/required interface) with an Event-B relation between two components and generates Domain and Range axioms for each interface. Figure 7 shows how to transform provided/required interface.



Sets
Constants R6.1
ObservedProperty R6.2
Monitoringdata R6.3
symptom R6.4
RFC R6.5
AdaptationActions
Commands R6.6
Axioms
ObservedProp_Relation:
ObservedProperty∈ContextElement→Probes R6.1
monidata_Relation: monitoringdata∈Probes↔{EC} R6.2
Symptom_Relation: symptom∈{SS}⇸{SR} R6.3
RFC_Relation: RFC∈{SR}⇸{RFCR}} R6.4
AdaAction_Relation: AdaptationActions∈{PS}⇸{PR} R6.5
Command_Relation: Commands∈{RE}→Effectors R6.6
dom(ObservedProperty)⊆ContextElement
ran(ObservedProperty)⊆Probes R6.1
dom(monitoringdata)⊆Probes
ran(monitoringdata)={EC} R6.2
dom(symptom)={SS}
ran(symptom)={SR} R6.3
dom(RFC)={SR}
ran(RFC)={RFCR} R6.4
……….

Fig. 7. Interface transformation rules

For example, the *ContextElement* and *Probe* communicate through *ObservedProperty* interface. This interface is provided by the *ContextElement* component and required by the *Probe*.

R6.1. This rule defines the graphical connection with an Event-B relation between *ContextElement* and *Probe* components and generates Domain and Range for ObservedProperty interface: $\mathrm{ObservedProperty} \in \mathrm{ContextElement} \rightarrow \mathrm{Probes}$

$\mathrm{dom(ObservedProperty)} \subseteq \mathrm{ContextElement}$

$\mathrm{ran(ObservedProperty)} \subseteq \mathrm{Probes}$

1.2 Behavioral features

We model the dynamic part of the Centralized pattern, with the Event-B method in the *MACHINE* part. Some variables defined in the graphical model are translated to invariants that define them. For example, the variable *PState* is defined with the invariant *Prob_Relation* (*PState ∈SensorNodes→ProbState)* which specify that *PState* is a relation between *SensorNodes* and *ProbState.*

Besides, the variable *WSNState* is defined with the invariant *WSNState_Relation (WSNState∈SinkNode→SymptomAnalysis)* which specify that *WSNState* is a relation between *SinkNode* and *SymptomAnalysis*. We transform each component of the MAPE control loop to an Event-B event. Each component name is transformed to an Event_name.

In fact, Models are developed in stepwise manners which are then automatically translated into Event-B specifications. In the first level of specification, we create a very abstract model: M0. In the next levels, we introduce gradually details and complexity into our model until obtaining the final pattern specification. Consequently, we refine a machine by adding new variables, strengthening the guards of an event or adding new guards and actions. Figure 8 shows an excerpt of the final Event-B specification of the Event monitor and analyze.

A machine is consistent if all its events preserve the invariants. The resulting Event-B specifications are enriched by invariants describing relevant properties like resiliency, safety, etc. that will be proved using the Rodin tool that generates Proof Obligations belonging to Event-B models.

```
Event Monitor
 ANY
   sensor
   data
   WHERE
   grd1:  sensor∈SensorNodes
   grd2:  TaskME(sensor)=Monitor
   grd3:  EvtColl(sensor)=Enabled
   grd4:  data∈ContextElement
   THEN
   act1:  TaskME(sensor):=Execute
   act2:  TaskAM:=Analyze
   act3:  EvtColl(sensor):=Disabled
   act4:  SenMo(sensor):=SS
   act5:  SensExe(sensor):=PR
   END
```

```
Event Analyze
   ANY
   sk
   WHERE
   grd1:  sk∈SinkNode
   grd2:  TaskAM=Analyze
   grd3:  WSNState(sk)=Degradation
   grd4:  SinkAna(sk)=SR
   THEN
   act1:  TaskAM:=Plan
   act2:  RequestForChange:=TRUE
   act3:  SinkAna(sk):=RFCS
   act4:  SinkPlan(sk):=RFCR
   END
```

Fig. 8. Excerpt of Event-B specification (Behavioral features)