1. SOL Language Specifications

1. Identify service and SAW information to be specified as inputs to the proof system
2. Define the specification format

2. Integration of SAW agents and workflow agents

(functionalities and runtime communication)

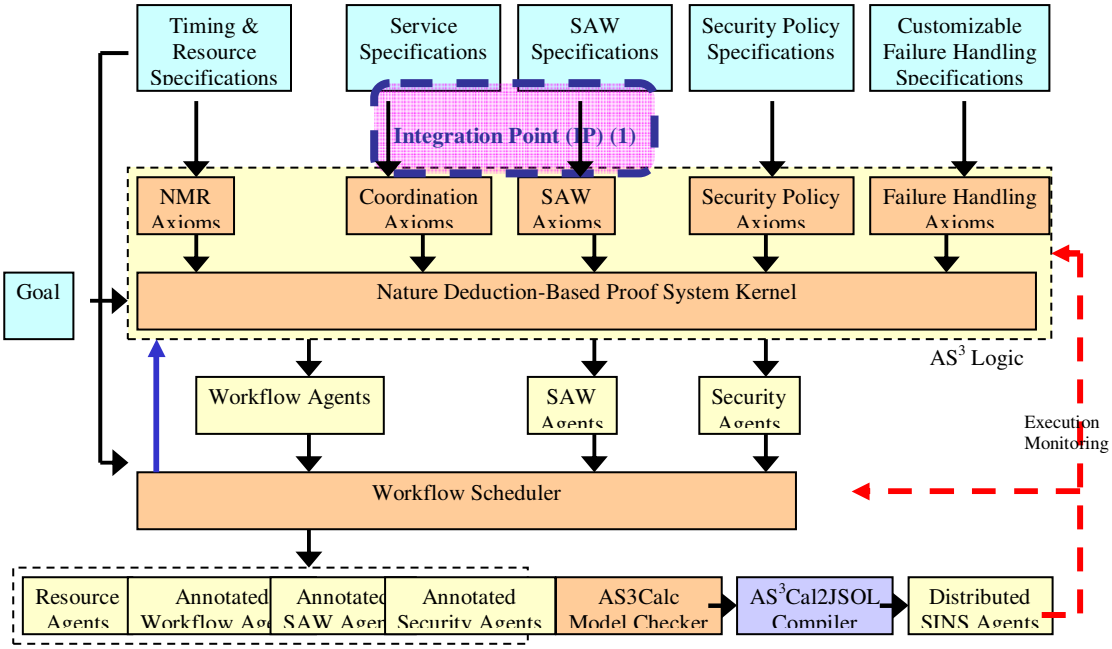


Figure 1. Our Approach to Rapid Development of AS^3 Systems

Figure 1 illustrates our approach to rapid development of AS^3 systems. In our approach, the service specifications and SAW specifications are generated by the SAW GUI tool and serve as input to the proof system. These specifications will be reasoned by the proof systems using the coordination axioms and saw axioms to generate workflow agents and SAW agents.

The distributed SAW information should be **reusable** by **multiple workflows** and distributed **security policies** and used for triggering **reactive behaviors** of the system. SAW agents are software components that responsible for collecting context data, analyzing situations, providing situational information to workflow agents and security agents and triggering workflow-independent actions when necessary. The functionalities of SAW agents and workflow agents are summarized as follows:

- <u>SAW agents are responsible for</u>
  a) Collect contexts by invoking sensing actions
  b) Receive results of goal-dependent actions from workflow agents when necessary
  c) Analyze situations
  d) Share situational information among different SAW agents when necessary

e) Broad cast situational information
f) Trigger goad-independent actions for system's reactive behaviors

❖ <u>Workflow agents are responsible for</u>
a) Be aware of preconditions of actions
b) Be aware of effects of goal-independent actions, which are triggered by SAW agents
c) Receive situational information from SAW agents, and invoke goal-dependent actions accordingly
d) Broad cast results of goal-dependent actions

Figure 2 describes the runtime communication between SAW agents and workflow agents using our ***Accident Response*** scenario.

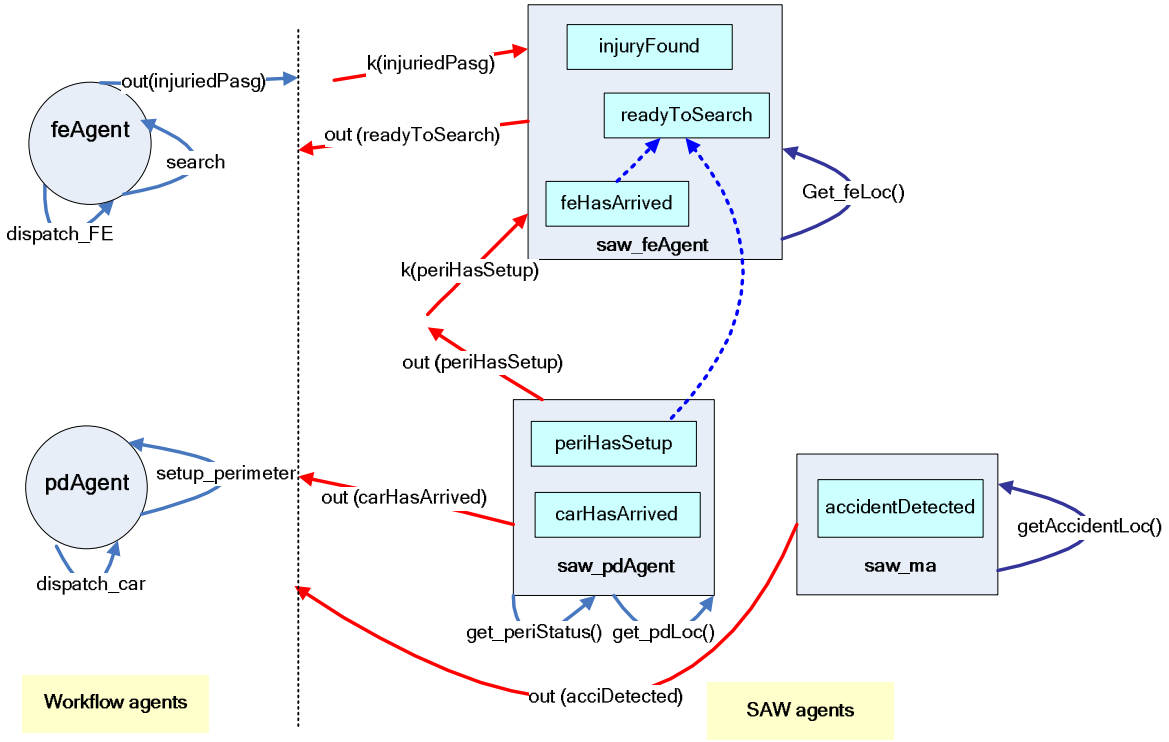


Figure 2. Runtime communication between workflow agents and SAW agents

Standard for naming channels:

1) Use situation name for sending and receiving situational information
2) Use service name concatenate method name (*e.g. fe_getInjuryStatus*) for sending and receiving action results

3. Identify information to be specified for the proof system

For coordination axioms

1) service specification
2) relation specification

For SAW axioms:

1) service specification
2) SAW specification
   a. Atomic situation specification
   b. Composite situation specification
   c. Trigger relation specification

4. Define Specification Format

❖ **<u>Service specification:</u>**

```
<service spec> ::=  <method> '=>>' <service>

<method> ::= <methodName> '(' <typed input list> ',' '<typed output list>' ',' <service name> ',' <agent name> ')'
<service> ::= 'serv' '(' <typed input list> ',' <typed output list>','<service name>','<agent name> ')'
<typed input list> | <typed output list> ::= '[ ]' | '['<typed item> ',' <typed item> … ']'
<typed item> ::= <typed variable> | <typed constant>
<typed variable> ::= <value type> '('<context type> '(' <variable> ')' ')'
<typed constant> ::= <value type> '(' <constant> ')'
<value type> := 'str' | 'int' | 'float' | 'bool'
<constant> ::= <string> | <integer> | <float> | <bool>
<variable>|<service name> | <agent name> ::= <lower-case-letter-started string>
```

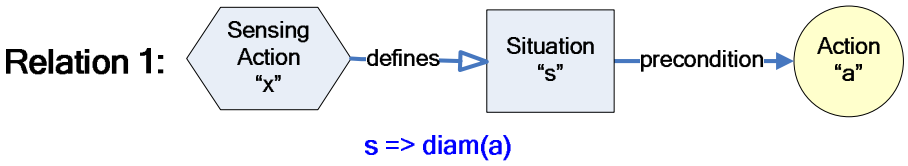
Example:

```
getAccidentLoc([],[int(locationType(r0))],cc911,saw_ma) =>>
          serv([],[int(locationType(r0))],cc911,saw_ma).
```

❖ **<u>Relation specification:</u>**

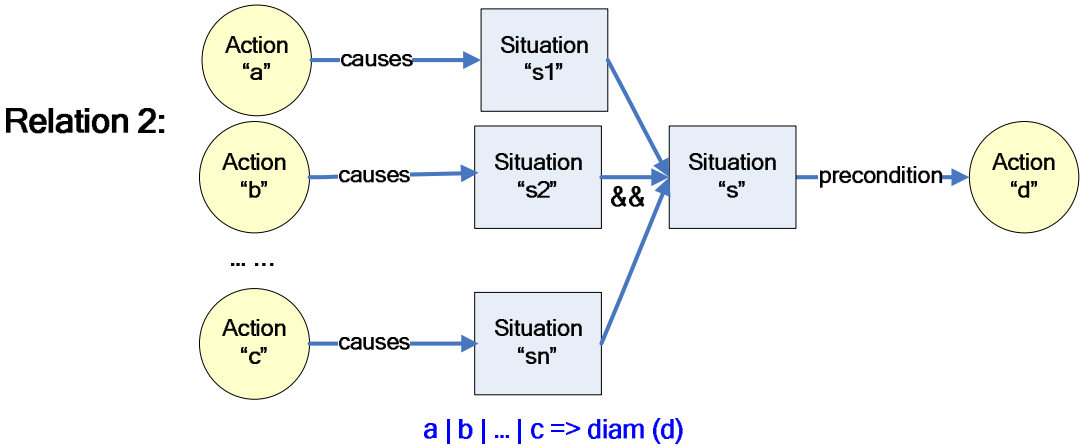
```
<relation spec> ::=
        % situation is precondition of an action
 (1)     <situation> '=>' 'diam(' <service> ')'

        % action, …, action => diam(action)
(2)     | '[' <serviceList> ']' '=>' 'diam(' <service> ')'

        % action, …, action  & situation => diam(action)
(3)     |  '[' <serviceList> ']' '=>' 'diam (if_then_else('<situation>, <sequenceOfServices>,<else condition>'))'
```

```
        % before
(4)     | '[' <serviceList> ']'  '=>' 'diam(' <service> '<<<' <service> ')'

<situation> ::= 'situation(' <typed output list>, <situation name> ')'

<serviceList> ::=
      <service>  | <service> ',' <serviceList>    % services can be executed in any order

<service> ::= 'serv(' <typed input list> ',' <typed output list>','<service name>','<agent name> ')'

<else condition> ::= 'failure' | 'true' | <service>| <sequenceOfServices>|<if_then_else>

<sequenceOfServices> = '<<<([' <serviceList> '])'
```



Example:

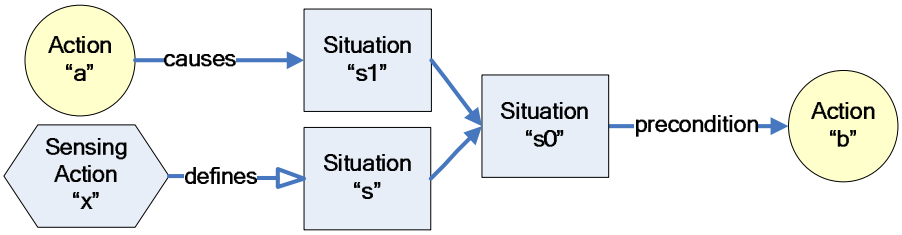
```
% accidentDetected => dispatchCar

situation([int(locationType(a0))], accidentDetected) =>
diam(serv([int(locationType(a0))],[str(carInfoType(r0))],car, pdAgent)).
```



Example:

```
% setupPerimeter() && dispatchFE()=> search()

  [serv([int(locationType(a0))],[str(periStatType(r0))],car, pdAgent),
   serv([int(locationType(a1))],[str(feInfoType(r1))],fe,feAgent)]
=> diam(serv([int(locationType(a2))],[str(ID(r2))],fe,feAgent)).
```

Relation 3:

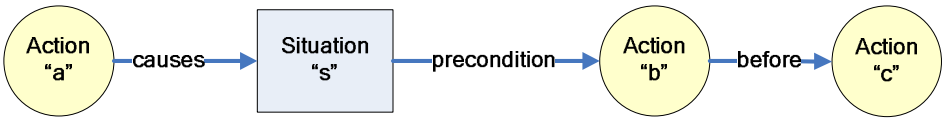


a && s => diam (b)

Example:

```
% dispatchCar() && visibilityLow => setupPerimeter()

[serv([int(locationType(a0))],[str(carInfoType(r0))],car, pdAgent)]
=> diam(if_then_else(
            situation([int(locatinType(a1))],visibilityLow),
            serv(int(locationType(a2)),str(periStatType(r1)),car, pdAgent),
            failure
         )

    ).
```

Relation 4:



a => diam (b <<< c)

Example:

```
% get_injuryStatus()=> dispatch_heli() <<< heli_sendToHospital()

[serv([int(passengerID(a0))],[str(passengerStatus(r0))],amb, ambAgent)]
=> diam(<<<([serv([int(locationType(a0))],[str(resultType(r1))],heli, heliAgent)
      serv([int(locationType(a0))],[str(resultType(r2))],heli, heliAgent)])).
```

❖ <u>Situation specification:</u>

<SAW spec> ::= <atomic situation> | <composite situation> | <trigger relation>

<atomic situation> ::=
'[' <atomic situation head list> ']' '= =>' 'diam ('<situation tail> ').'

<atomic situation head list> ::= <atomic situation head> | <atomic situation head> ',' <atomic situation head list>

<atomic situation head> ::=
<service> | <know> | <compare>

<know > ::=
'k'(' '<typed output list> ',' <situation name> ',' <agent name> ')'
| 'k'(' '<typed output list> ',' <service name> '_'<method name> ',' <agent name> ')'

<compare> ::= 'compare('<variable> ',' <operator> ',' <const> ')'

```
<situation tail> ::= ‘k(’ <typed output list>‘,’ <situation name>‘,’ <monitor_until> ‘,’ <agent name>‘)’

<composite situation> ::= <composite situation head> ‘= =>’ ‘diam k(’<situation tail> ‘)’

<composite situation head > ::=  < situation tail>
                                  |  ‘and(‘ <composite situation head> ‘,’ <composite situation head> ‘)’
                                  |  ‘or(’ <composite situation head> ‘,’ <composite situation head> ‘)’

<monitor_until> ::= ‘monitor_until(’ <frequency> ‘,’ <termination> ‘,’ <reinitialization> ‘)’

<frequency> ::= <int constant>

<termination> ::= <situation name>

<reinitialization> ::= <Boolean constant>

<trigger relation> ::=  ‘trigger(’ <situation tail >, <service> ‘)’
```

Example:

```
** atomic situation -- feHasArrived **

[k([int(accidentLoc)],accidentDetected, feHasArrived_Agent),
 serv([int(feIDType(feID))],[int(locationType(feLoc))],fe,feHasArrived_Agent),
 compare(accidentLoc, '==', feLoc)]
==> diam(k([int(accidentLoc),int(feLoc)], feHasArrived, monitor_until(20, rescueSuccess, true),
        feHasArrived_Agent)).

** composite situation -- readyToSearch **

[or(and( k([int(accidentLoc), int(feLoc)], feHasArrived, readyToSearch_Agent),
        k([str(periStatus)], periHasSetuped, readyToSearch_Agent)
       ),
    k([int(accidentLoc), int(pdLoc)], pdHasArrived, readyToSearch_Agent)
   )]
==> diam(k([int(accidentLoc), int(feLoc), str(periStatus)], readyToSearch, monitor_until(100, rescueSuccess,
        true), readyToSearch_Agent)).


** atomic situation -- injuryFound **

[k([str(passengerID)],fe_search, injuryFound_Agent),
 compare(passengerID, '!=', """)]
==> diam(k([str(passengerID)], injuryFound, monitor_until(-1, null, no), injuryFound_Agent)).
```

5. Tasks to be done for providing the specifications

Add new functionalities to SAW GUI tool for:

1) specify ‘before’ relation among goal-dependent actions
2) differentiate sensing actions, goal-independent actions and goal-dependent actions

Modify existing functionalities of SAW GUI tool for:

3) generate action dependency specifications
4) generate situation specification
5) generate trigger relation specification