Copland Core Language Definition and JSON Message Schemas (An extension of MITRE/KU/APL Copland language)

Adam Petz and Perry Alexander

University of Kansas

1 Terms(t) and Evidence Types (E)

$$
\begin{aligned}
P &\leftarrow place \\
M &\leftarrow asp\_id \\
A &\leftarrow \mathsf{USM}\ M\ \bar{a} \mid \mathsf{KIM}\ M\ P\ \bar{a} \mid \mathsf{SIG} \mid \mathsf{HSH} \mid \mathsf{CPY} \mid \cdots \\
t &\leftarrow A \mid @_P\, t \mid (t \to t) \mid (t \overset{\pi}{\prec} t) \mid (t \overset{\pi}{\sim} t) \\
E &\leftarrow \xi \mid \mathsf{U}_P(E) \mid \mathsf{K}_P^P(E) \mid [\![E]\!]_P \mid \#_P\, E \mid \mathsf{N}_P(E) \mid (E\ ;;\ E) \mid (E \parallel E) \mid \cdots
\end{aligned}
$$

$\pi = (\pi_1, \pi_2)$ is a pair of evidence splitting functions
$\bar{a}$ is a list of string arguments
$place$, $asp\_id \in \mathbb{N}$

Fig. 1. Term Grammar

2 Concrete Evidence(e)

$$
\begin{aligned}
\mathsf{BS} &\leftarrow bits \\
e &\leftarrow \mathsf{mt} \mid \mathsf{U}_P\ M\ \bar{a}\ \mathsf{BS}\ (e) \mid \mathsf{K}_P^P\ M\ \bar{a}\ \mathsf{BS}\ (e) \mid \mathsf{G}_P\ e\ \mathsf{BS} \mid \mathsf{H}_P\ \mathsf{BS} \mid \mathsf{N}_P\ \mathsf{BS}\ (e) \mid \mathsf{SS}\ e\ e \mid \mathsf{PP}\ e\ e \cdots
\end{aligned}
$$

Fig. 2. Conrete evidence Grammar

3 Messages

$$
\begin{aligned}
&RequestMessage = \{ \\
&\quad toPlace :: P, \\
&\quad fromPlace :: P, \\
&\quad reqNameMap :: P => Address, \\
&\quad reqTerm :: t, \\
&\quad reqEv :: e \\
&\quad \} \\
&ResponseMessage = \{ \\
&\quad respToPlace :: P, \\
&\quad respFromPlace :: P, \\
&\quad respEv :: e \\
&\quad \}
\end{aligned}
$$

Fig. 3. Request and Response Message record structures

4 Data Exchange Format (JSON Schema)

4.1 Message Schemas

We represent Request and Response Messages as record structures (Figure 3). Their corresponding JSON object schemas are as follows:

RequestMessage Schema

```
{
  "toPlace": < number >,
  "fromPlace": < number >,
  "reqNameMap": < nameMap >,
  "reqTerm": < term >,
  "reqEv": < evidence >
}
```

ResponseMessage Schema

```
{
  "respToPlace": < number >,
  "respFromPlace": < number >,
  "respEv": < evidence >
}
```

< number > is the JSON primitive number type. < term > and < evidence > are JSON objects for Copland terms and evidence respectively, and are defined below.
< nameMap > is a JSON object of the following form:
$\{pl_1{:}addr_1, pl_2{:}addr_2, ...\}$
where pl_1, pl_2, ... are JSON key strings that represent a Copland place identifier and $addr_1$, $addr_2$, ... are JSON strings that represent platform addresses (we leave address strings abstract in this specification, but a common example would be a string of the form ip:port).

4.2 General ADT Schema

We represent Copland terms (t in Figure 1) and concrete evidence (e in Figure 2) as Algebraic Data Types (ADTs). Every JSON object representing an ADT has two members:

1. "constructor"-maps to the constructor name string (e.g. "KIM", "K", "AT"). Note: Constructor names should be unique to allow unambiguous parsing.
2. "data"-maps to an ***ordered*** JSON array that holds the arguments for that particular constructor(members of the data array will differ from constructor to constructor).

```
{
  "name": < string >,
  "data": < array >
}
```

< string > and < array > are the JSON primitive string and array types respectively.

4.3 Copland Term Constructor Schemas

Corresponds to the constructors of the Copland term grammar (t) in Figure 1. These object schemas satisfy the < term > placeholder.

```
{
  "name": "USM",
  "data": [
    < number >,
    [< string >]
    ]
}
```

```
{
  "name": "KIM",
  "data": [
    < number >,
    < number >,
    [< string >]
  ]
}
```

```
{
  "name": "SIG"
}
```

```
{
  "name": "HSH"
}
```

```
{
  "name": "CPY"
}
```

```
{
  "name": "AT",
  "data": [
    < number >,
    < term >
  ]
}
```

```
{
  "name": "LN",
  "data": [
  < term >,
  < term >
  ]
}
```

```
{
  "name": "BRS",
  "data": [
    [<"ALL" | "NONE">, <"ALL" | "NONE">],
    < term >,
    < term >
  ]
}
```

```
{
  "name": "BRP",
  "data": [
    [<"ALL" | "NONE">, <"ALL" | "NONE">],
    < term >,
    < term >
  ]
}
```

Note: the notation <"ALL" | "NONE"> means "one of the "ALL" or "NONE" evidence splitting functions". "ALL" and "NONE" are simply JSON string constants. The order of these strings in the subarray is significant-the first element of the array corresponds to the first < term > subterm, and similarly for the second.

4.4 Concrete Evidence Constructor Schemas

Corresponds to the constructors of the concrete evidence grammar (*e*) in Figure 2. These object schemas satisfy the < evidence > placeholder.

Note: Values for fields that hold "bits" should be standard base16-encoded strings (representing arbirary binary data–hashes, nonces, signatures, etc.).

```
{
  "name": "U",
  "data": [
    < number >,
    [< string >],
    < number >,
    < string >,
    < evidence >
  ]
}
```

```
{
  "name": "K",
  "data": [
    < number >,
    [< string >],
    < number >,
    < number >,
    < string >,
    < evidence >
  ]
}
```

```
{
  "name": "G",
  "data": [
    < number >,
    < evidence >,
    < string >
  ]
}
```

```
{
  "name": "H",
  "data": [
    < number >,
    < string >
  ]
}
```

```
{
  "name": "N",
  "data": [
    < number >,
    < string >,
    < evidence >
  ]
}
```

```
{
  "name": "SS",
  "data": [
  < evidence >,
  < evidence >
  ]
}
```

```
{
  "name": "PP",
  "data": [
  < evidence >,
  < evidence >
  ]
}
```