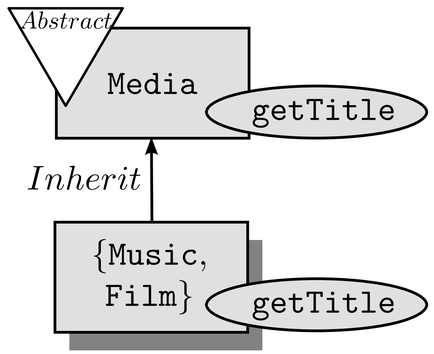
Linking Codecharts with Programs

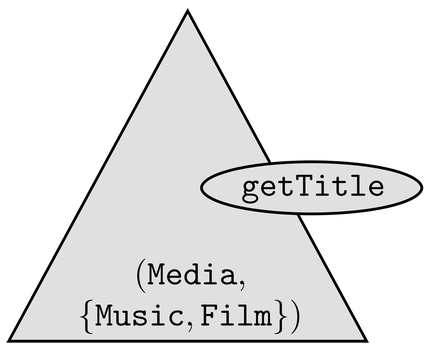
Jon Nicholson, Aidan Delaney and Gem Stapleton
Visual Modelling Group, University of Brighton

Concrete Codecharts

Codecharts [1] are a formal visual language for specifying the structure of object-oriented programs, frameworks and design patterns. Their vocabulary represents common concepts in statically typed class based object-oriented program design: classes, methods, signatures, properties and relationships.



In the above, class Media is ***Abstract***, classes Music and Film both ***Inherit*** from Media, and these classes all have a method with the signature getTitle. The below specifies the same design decisions except that Media is not explicitly specified as ***Abstract***.



In [2] we formalized this concrete syntax, thereby providing means to decide the syntactic well-formedness of codecharts.

Abstract Codecharts

Concrete codecharts contain a lot of information that is not relevant to its semantics, e.g. size and layout. We prefer, therefore, to reason over an abstract syntax that discards such semantically unimportant geometric information.

An abstract codechart is a tuple composed of multisets (bags) containing labels and a function preserving semantically desirable topological information. Bags capture occurrences of labels such as to reason over them independently.

- $\mathcal{R} = \{(\texttt{Media}, 1)\}$
- $\mathcal{OR} = \{(\{\texttt{Music}, \texttt{Film}\}, 1)\}$
- $\mathcal{E} = \{(\texttt{getTitle}, 1), (\texttt{getTitle}, 2)\}$
- $\mathcal{T}_{\triangledown} = \{(\textit{Abstract}, 1)\}$
- $\mathcal{A}_{\blacktriangleright} = \{((\textit{Inherit}, (\{\texttt{Music}, \texttt{Film}\}, 1), (\texttt{Media}, 1)), 1)\}$
- $\omega((\texttt{getTitle}, 1)) = (\texttt{Media}, 1)$,
  $\omega((\texttt{getTitle}, 2)) = (\{\texttt{Music}, \texttt{Film}\}, 1)$,
  $\omega((\textit{Abstract}, 1)) = (\texttt{Media}, 1)$

Above and below are the abstract codecharts for our examples.

- $\mathcal{T}_{\triangle} = \{((\texttt{Media}, \{\texttt{Music}, \texttt{Film}\}), 1)\}$
- $\mathcal{E} = \{(\texttt{getTitle}, 1)\}$
- $\omega((\texttt{getTitle}, 1)) = ((\texttt{Media}, \{\texttt{Music}, \texttt{Film}\}), 1)$

Our paper describes, at the conceptual level, how our abstract syntax help to link codecharts with programs. For example, showing that both the Java and C++ programs (right) implement both codecharts.

Programs

The semantics of codecharts are based on OO programming languages and therefore requires a mapping to OO concepts be defined. For example, to map the class name Media to the Java class Media, ***Abstract*** to the keyword **abstract**, and ***Inherit*** to the keywords **extends** and **implements**.

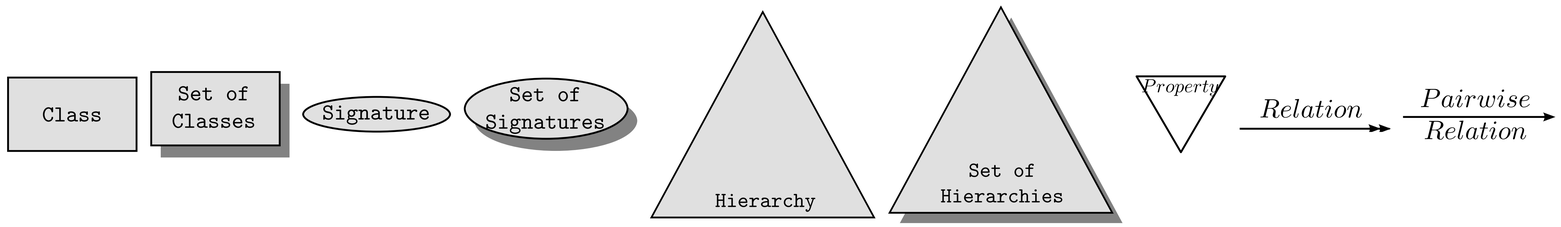
```
1 abstract class Media { ...
2   public abstract String getTitle(); }
3 class Music extends Media { ...
4   public String getTitle() { ... } }
5 class Film extends Media { ...
6   public String getTitle() { ... } }
```

Other OO languages can be used, such as C++ (below), if such a mapping can be defined for it. For example, Media is ***Abstract*** as it contains a pure virtual method.

```
1 class Media { ...
2   public: virtual string getTitle()=0; };
3 class Music : public Media { ...
4   public: string getTitle() { ... } };
5 class Film : public Media { ...
6   public: string getTitle() { ... } };
```

Future work includes defining a model theoretic semantics and to investigate applications such as visualizing program metrics.

Syntax Reference



References

[1] A.H. Eden and J. Nicholson.
Codecharts: Roadmaps and Blueprints for OO Programs.
Wiley-Blackwell, April 2011.

[2] J. Nicholson, A. Delaney, and G. Stapleton.
Formalizing the syntax of codecharts.
In *Proc. of DMS 2012 (VLC 2012)*, Miami, USA, August 2012.