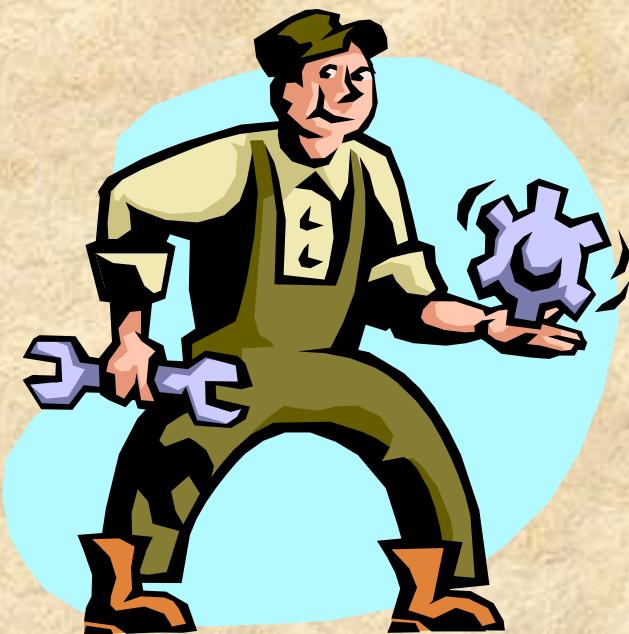


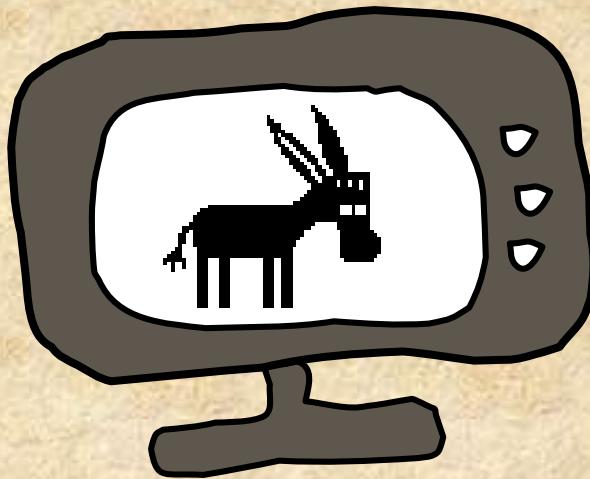


ΗΥ454 : ΑΝΑΠΤΥΞΗ ΕΞΥΠΝΩΝ ΔΙΕΠΑΦΩΝ ΚΑΙ ΠΑΙΧΝΙΔΙΩΝ

**ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ,
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ,
ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ**



**ΔΙΔΑΣΚΟΝΤΕΣ
Αντώνιος Σαββίδης**



**ΕΞΥΠΝΕΣ ΔΙΕΠΑΦΕΣ,
Σύνολο διαλέξεων 6, Διάλεξη 2η**



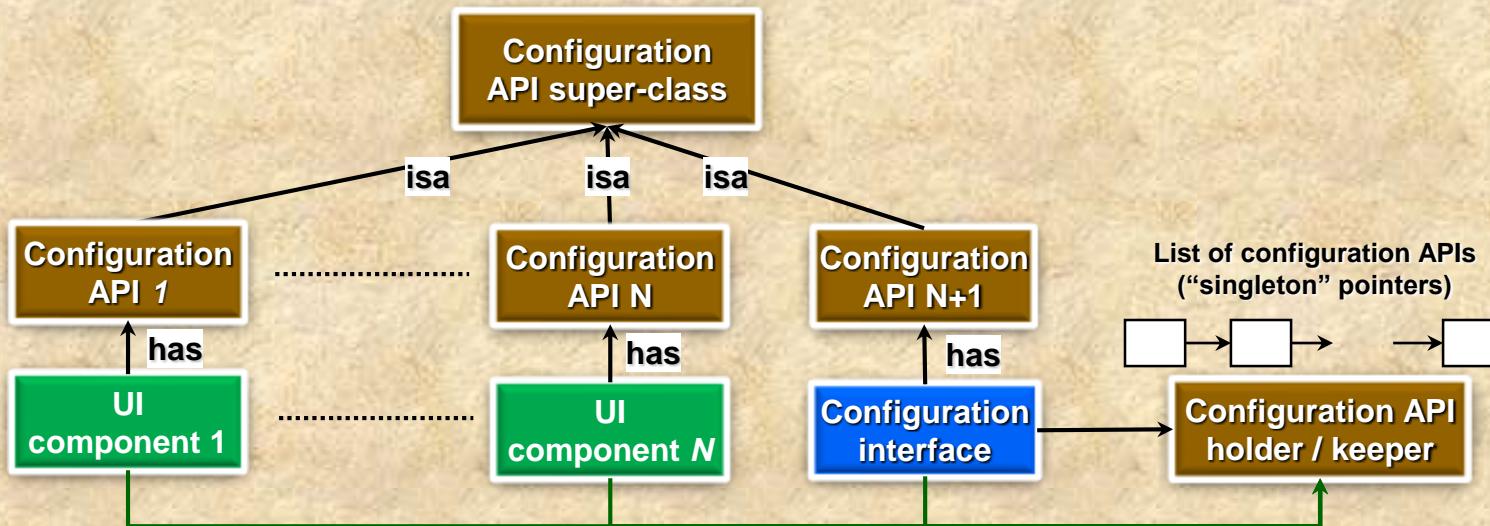
Περιεχόμενα

- **Υλοποίηση διαλογικών εργαλείων προσαρμογής**
- Ενσωματωμένοι διάλογοι προσαρμογής
- Προσαρμογές με scripting languages
- Επίλογος



Διαλογικά εργαλεία I (1/8)

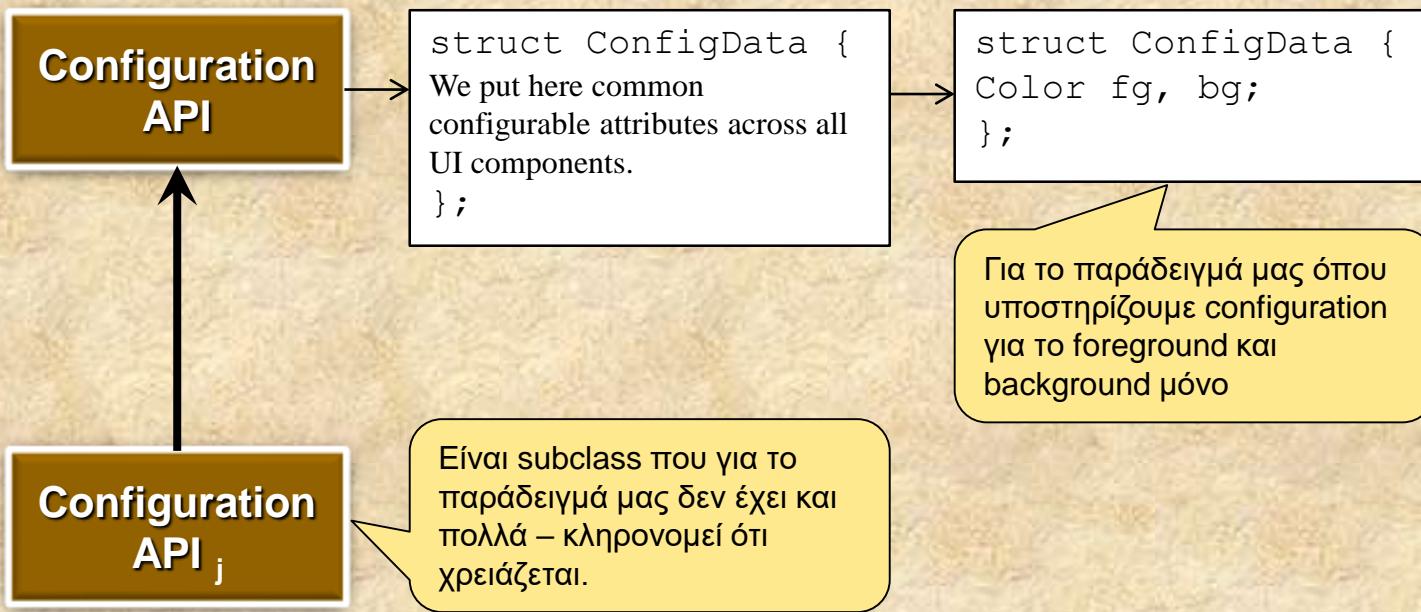
■ Η γενική αρχιτεκτονική της υλοποίησης



- Τα configuration APIs είναι singleton classes (HY352, lecture 14), τα οποία γίνονται registered στην singleton κλάση “API holder” από τα αντίστοιχα UI components (κεντρική κλάση ανά component), μέσα στην συνάρτηση αρχικοποίησης τους.
- Το configuration interface είναι και αυτό ένα τμήμα της διεπαφής της εφαρμογής, το οποίο ουσιαστικά προσφέρει ένα user interface για το configuration API (θα εξηγηθεί καλύτερα με ένα παράδειγμα).



Διαλογικά εργαλεία I (2/8)

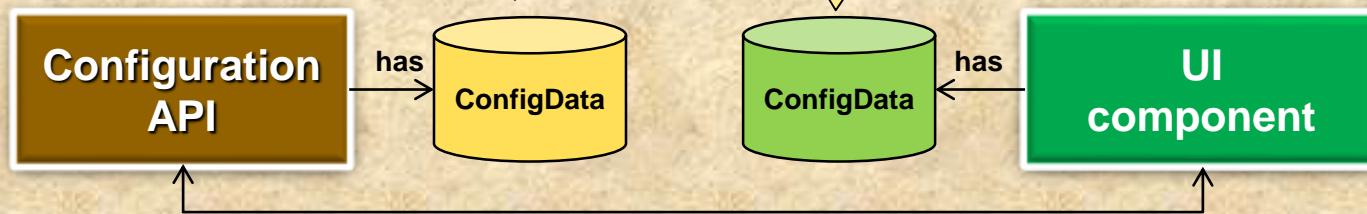




Διαλογικά εργαλεία I (3/8)

Temp, χρησιμοποιούνται κατά τη διάρκεια του configuration.

Είναι αυτά που χρησιμοποιούνται από το UI component.



```
// Abstract class (interface) for all UI components
class UIComponent {
protected:
    ConfigData configData; // editor display attrs
public:
    virtual void Draw (const ConfigData& data) const = 0;
    virtual ConfigAPI& GetConfigAPI (void) const = 0;
    virtual const string GetId (void) const = 0;
    void Preview (void)
        { Draw(GetConfigAPI().GetConfigData()); }
    void Start (void) // read UI component attrs
        { GetConfigAPI().GetConfigData() = configData; }
    void Cancel (void) // redraw with original UI component attrs
        { Draw(configData); }
    void Apply (void) // write UI component attrs
        { Draw(configData = GetConfigAPI().GetConfigData()); }
};
```



Διαλογικά εργαλεία I (4/8)

```
class ConfigAPI {
public:
protected:
    UIComponent* ui;      // linkage to its UI component
    ConfigData configData;
public:
    void SetForeground (const Color& c)
        { configData.fg = c; }
    const Color GetForeground (void) const
        { return configData.fg; }
    void SetBackground (const Color& c)
        { configData.bg = c; }
    const Color GetBackground (void) const
        { return configData.bg; }
    const ConfigData&
        GetConfigData (void) const
        { return configData; }
    virtual const string
        GetId (void) const = 0;
    UIComponent* GetUI (void)
        { return ui; }
    void SetUI (UIComponent* _ui)
        { ui = _ui; }
    ConfigAPI (void) { ui = NULL; }
};
```

Methods για τα configuration data

Methods για σύνδεση με UI components



Διαλογικά εργαλεία I (5/8)

```
// Singleton class (single instance) to hold apis
class ConfigAPIs {
private:
    static list<ConfigAPI*> apis;
public:
    static void Add (ConfigAPI* api)
    { apis->push_back(api); }
    static list<ConfigAPI*>& Get (void)
    { return apis; }
};
```



Διαλογικά εργαλεία I (6/8)

```
class Editor : public UIComponent {
    class EditorConfigAPI : public ConfigAPI {
        public:
            const string     GetId (void) const { return "Editor"; }
    };
protected:
    static EditorConfigAPI  configAPI;
public:
    ConfigAPI&           GetConfigAPI (void) const
                            { return configAPI; }
    void                 Draw (const ConfigData& data) const
                            { /* draw using fg, bg from 'data' */ }
    const std::string    GetId (void)
                            { return "Editor"; }

    static void           Initialise (void)
                            { ConfigAPIs::Add(&configAPI); }

    Editor (void) { configAPI.SetUI(this); }
    ~Editor (void) { configAPI.SetUI((UIComponent*) 0); }
};
```



Διαλογικά εργαλεία I (7/8)

■ Ένα απλό παράδειγμα (1/2)

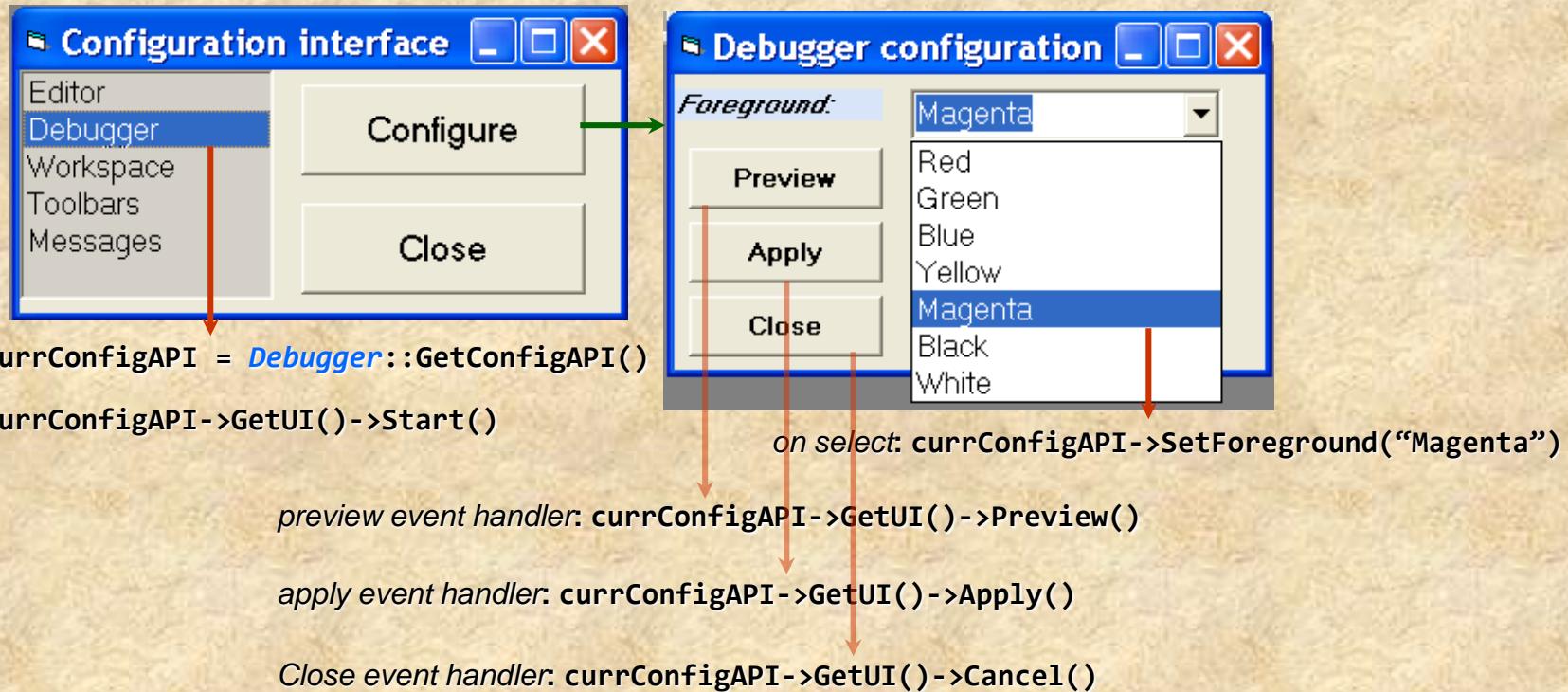
- Τι υλοποιεί λοιπόν το *configuration interface*?
 - ◆ «Παίρνει» τη λίστα με τα configuration API instances καλώντας την **ConfigAPIs::Get()**
 - ◆ Κάνοντας iterate στη λίστα δημιουργεί ένα menu με τα ονόματα όλων των διαθέσιμων configuration APIs (καλώντας την **GetId**)
 - ◆ Δημιουργεί μία «φόρμα» για την επιλογή του foreground color δίνοντας τη δυνατότητα για preview και apply για το εκάστοτε configuration API
 - χρησιμοποιεί τις **GetUI->Start, Preview, Cancel** και **Apply()** αναλόγως - εάν η **GetUI** δεν επιστρέψει null

➔ Γενικά η όλη τεχνική του διαχωρισμού μεταξύ *configuration interface*, *User Interface component* και *configuration API* δανείζεται χαρακτηριστικά από το *View pattern – HY352, Lecture 16*.



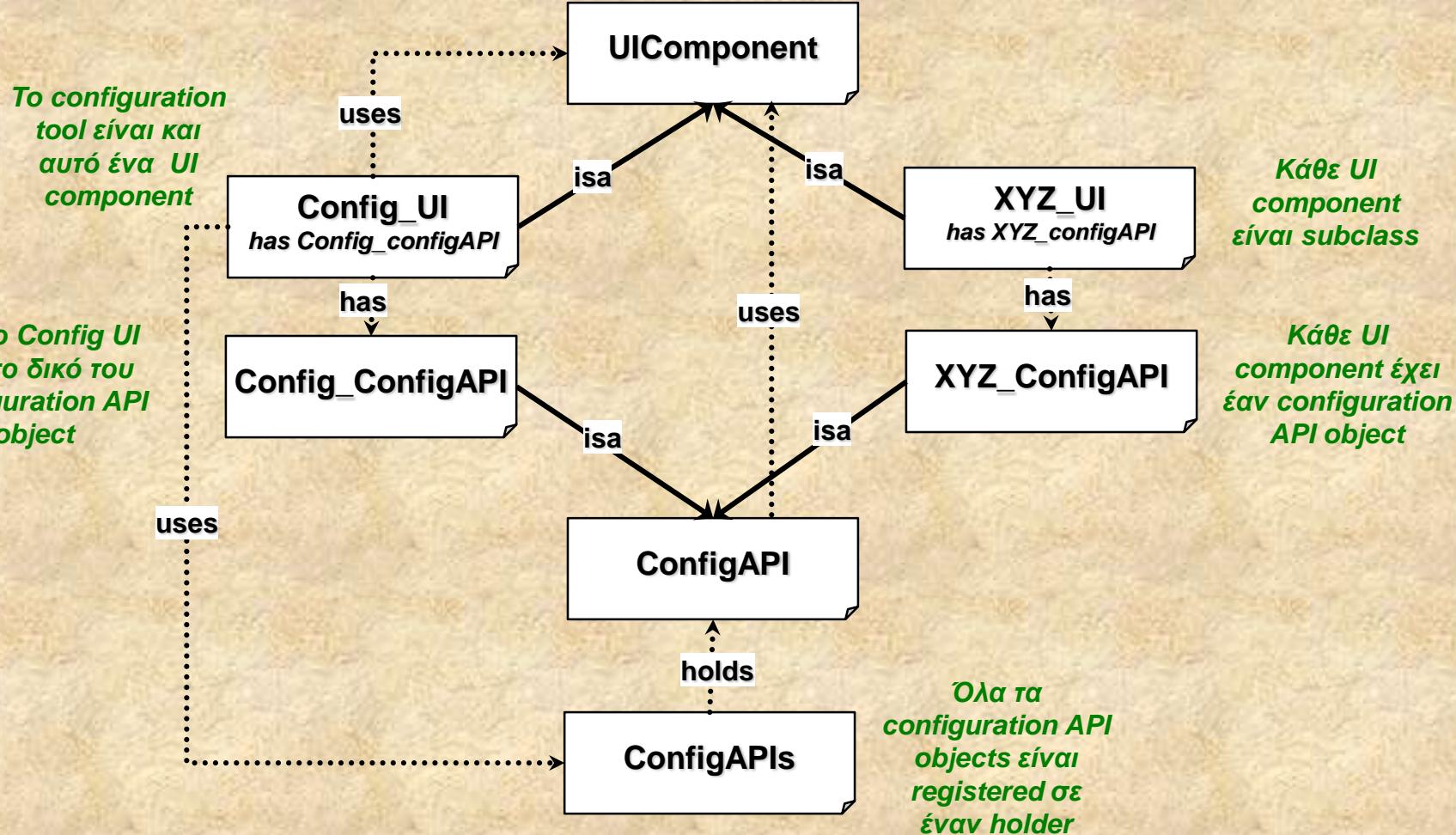
Διαλογικά εργαλεία I (8/8)

■ Ένα απλό παράδειγμα (2/2)





Ένθετο





Διαλογικά εργαλεία II (1/15)

- Στην πρώτη υλοποίηση όλα τα UI components έχουν ακριβώς τα ίδια προσαρμόσιμα χαρακτηριστικά
- Κάτι τέτοιο είναι σπάνιο, ωστόσο αυτό το sharing ήταν που μας έδωσε τις εξής δυνατότητες:
 - Να μεταφέρουμε αρκετό κώδικα στο *ConfigAPI* super-class
 - Να κάνουμε τα derived *XZY_ConfigAPI* classes πολύ απλά
 - Να αυτοματοποιήσουμε την κατασκευή του *ConfigAPI* UI
- Το πρόβλημα μας ανάγεται στο εξής:
 - Πως μπορούμε να μοντελοποιήσουμε τα configuration data ούτως ώστε με μία κλάση να καλύπτουμε τις ανάγκες όλων των UI components
 - Έτσι θα αντικαταστήσουμε το *ConfigData* με αυτή την κλάση και θα συνεχίσουμε να έχουμε sharing
 - *Ιδέες;*



Διαλογικά εργαλεία II (2/15)

- Η τεχνική βασίζεται στην τυποποίηση των τύπων των προσαρμόσιμων χαρακτηριστικών σε ένα μικρό σύνολο από κλάσεις
 - και στον ορισμό κατάλληλων μικρό-διεπαφών για κάθε κλάση ώστε να μπορούν να γίνονται edited τα χαρακτηριστικά από τον χρήστη
- ➔ μία τέτοια περίπτωση είδαμε στο προηγούμενο παράδειγμα:
 - ➔ Προσαρμόσιμο χαρακτηριστικό «color»
 - ➔ το οποίο αντιστοιχίστηκε σε τύπο «λίστα από strings»,
 - ➔ με μικρό-διεπαφή ένα «combo box».



Διαλογικά εργαλεία II (3/15)

```
class Property {
protected:
    string           id;
public:
    enum Type      {
        Enumerated,
        IntRange,
        Boolean,
        Numeric,
        Aggregate
    };
    virtual Property*   Clone (void) const = 0;
    virtual void        Assign (const Property& prop) = 0;
    virtual Type        GetType(void) const = 0;
    const string        GetId (void) const
                        { return id; }
    Property (const string& _id) : id(_id){}
    virtual ~Property(){}
};
```



Διαλογικά εργαλεία II (4/15)

```
class EnumeratedProperty : public Property {
protected:
    unsigned          index; // current selection
    vector<string>   domain; // value domain
public:
    const vector<string>&
                    GetDomain (void) const;
    unsigned          GetIndex (void) const
                    { return index; }
    void             SetIndex (unsigned i)
                    { index = i; }
    const string&   GetValue (void) const
                    { return domain[index]; }
    virtual Type    GetType (void) const
                    { return Property::Enumerated; }

    virtual Property* Clone (void) const {
        return new EnumeratedProperty(
            id, domain, index
        );
    }
    virtual void     Assign (const Property& prop) {
        assert(prop.GetType() == GetType()); // types match
        const EnumeratedProperty& e = (EnumeratedProperty&) prop;
        assert(e.domain == domain);           // domains identical
        index = e.index;
    }

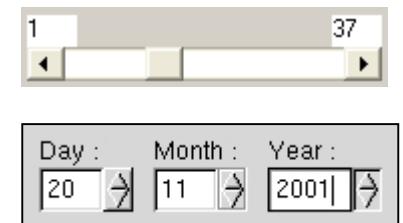
    EnumeratedProperty (const string& id, const vector<string>& d, unsigned i);
};
```

The screenshot shows a Windows-style dialog box titled "Enumerated". Inside the dialog, there is a list box containing five items: "One", "Two", "Three", "Four", and "Five". The item "Three" is currently selected, indicated by a blue background. Below the list box is a radio button group with five options labeled "One", "Two", "Three", "Four", and "Five". The radio button for "Two" is selected, indicated by a solid blue circle.



Διαλογικά εργαλεία II (5/15)

```
class IntRangeProperty : public Property {
protected:
    int left, right, step, value;
public:
    int GetLeft (void) const { return left; }
    int GetRight (void) const { return right; }
    int GetStep (void) const { return step; }
    int GetValue (void) const { return value; }
    void SetValue (int _value) { value = _value; }
    virtual Type GetType (void) const
        { return Property::IntRange; }
    virtual Property* Clone (void) const {
        return new IntRangeProperty(
            id, left, right, value
        );
    }
    virtual void Assign (const Property& prop) {
        assert(prop.GetType() == GetType()); // types match
        const IntRangeProperty& i = (IntRangeProperty&) prop;
        assert(i.left == left && i.right == right); // intervals identical
        value = i.value;
    }
    IntRangeProperty (const string& id, int l, int r, int v) :
        Property(id), left(l), right(r), value(v), step(1) {
        assert(left < right && step > 0);
        assert(left <= value && value <= right);
        assert(step < (right - left));
    }
};
```



1 37

Day : Month : Year :
20 ⏪ 11 ⏩ 2001 ⏩



Διαλογικά εργαλεία II (6/15)

Identifier

```
class BooleanProperty : public Property {
protected:
    bool             value;
public:
    bool            GetValue (void) const { return value; }
    void            SetValue (bool _value) { value = _value; }
    virtual Type    GetType (void) const
                    { return Property::Boolean; }
    virtual Property* Clone (void) const {
        return new BooleanProperty(id, value);
    }
    virtual void    Assign (const Property& prop) {
        assert(prop.GetType() == GetType()); // types match
        const BooleanProperty& b = (BooleanProperty&) prop;
        value = b.value;
    }
    BooleanProperty (const string& id, bool v) : Property(id), value(v) {}
};
```



Διαλογικά εργαλεία II (7/17)

34.234

```
class NumericProperty : public Property {
protected:
    float             value;
public:
    float            GetValue (void) const { return value; }
    void             SetValue (float _value) { value = _value; }
    virtual Type     GetType (void) const
                    { return Property::Numeric; }
    virtual Property* Clone (void) const {
        return new NumericProperty(id, value);
    }
    virtual void     Assign (const Property& prop) {
        assert(prop.GetType() == GetType()); // types match
        const NumericProperty& n = (NumericProperty&) prop;
        value = n.value;
    }
    NumericProperty (const string& id, float v) : Property(id), value(v) {}
};
```



Διαλογικά εργαλεία II (8/15)

```
class AggregateProperty : public Property {
public:
    // with iterator i, key: i->first, value: i->second
    typedef std::map<std::string, Property*> Properties;
protected:
    Properties          props;
public:
    virtual Type        GetType (void) const { return Property::Aggregate; }
    virtual Property*   Clone (void) const   { return new AggregateProperty(id, props); }
    virtual void         Assign (const Property& prop);

    const Properties&  Get (void) const      { return props; }
    void               Add (Property* p)     { props[p->GetId()] = p; }
    Property*          Get (const std::string& id) {
        Properties::iterator i = props.find(id);
        return i != props.end() ? i->second : (Property*) 0;
    }
    const Property*    Get (const std::string& id) const {
        Properties::const_iterator i = props.find(id);
        return i != props.end() ? i->second : (const Property*) 0;
    }
    void               Set (const Properties& _props); // clears and sets
    void               Clear (void);

AggregateProperty (const string& id) : Property(id) {}
AggregateProperty (const string& id, const Properties& _props) :
    Property(id) { Set(_props); }
virtual ~AggregateProperty(){ Clear(); }
};
```



Διαλογικά εργαλεία II (9/15)

```
void AggregateProperty::Assign (const Property& prop) {
    assert(prop.GetType() == GetType()); // types match
    const AggregateProperty& a = (AggregateProperty&) prop;

    // iterate and assign each property
    Properties::iterator i = props.begin();
    Properties::const_iterator j = a.props.begin();

    while (i != props.end()) {
        assert(j != a.props.end());
        i->second->Assign(*j->second);
        ++i, ++j;
    }
    assert(j == a.props.end());
}

void AggregateProperty::Set (const Properties& _props) {
    Clear();
    for (Properties::const_iterator i = _props.begin(); i != _props.end(); ++i)
        props[i->first] = i->second->Clone();
}

void AggregateProperty::Clear (void) {
    Properties::iterator i = props.begin();
    while (i != props.end()) {
        delete i->second; // delete property
        i = props.erase(i); // erases and returns next iterator value
    }
}
```



Διαλογικά εργαλεία II (10/15)

■ Τι έχουμε μέχρι αυτό το σημείο;

- Έχουμε ξεχωριστές κλάσεις για κάθε διαφορετικό «τύπο προσαρμόσιμης» παραμέτρου του User Interface
- Οι κλάσεις αυτές «μιμούνται» τις εγγενείς κλάσεις της γλώσσας, π.χ., *bool*, *int*, *string*, *list*, κλπ, αλλά με τρόπο που επιτρέπει να έχω έναν super type και δυνατότητα να ελέγχω τον πραγματικό τύπο με το *DerivedType*.
 - ◆ Μπορούμε να το αποφύγουμε εάν εφαρμόσουμε το visitor pattern
- Έχουμε δηλαδή ορίσει κάποια ειδικά είδη adapter / wrapper classes για βασικούς τύπους δεδομένων
- Για ένα Property τι μπορώ μα κάνω?
 - ◆ Να φτιάξω μία function η οποία ανάλογα με το derived class κατασκευάζει το αντίστοιχο micro interface
 - ◆ Μάλιστα, αυτά τα διάφορα micro interfaces μπορώ να τα υλοποιήσω και ως διαφορετικές κατάλληλες κλάσεις
- Μετατρέποντας τους τύπους σε τιμές του καθιστούμε **first-class values**, δηλ. δεδομένα σε αλγόριθμο



Διαλογικά εργαλεία II (11/15)

```
// a common config data type for all ui components
// and config apis
typedef AggregateProperty ConfigData;

// now a normal non-abstract class, same for all UI components
class ConfigAPI {
protected:
    UIComponent* ui; // linkage to its UI component
    ConfigData configData;

public:
    ConfigData& GetData (void)
    { return configData; }
    const ConfigData& GetData (void) const
    { return configData; }
    const string GetId (void) const // same id as its ui component
    { assert(ui); return ui->GetId(); }
    UIComponent* GetUI (void) { return ui; }
    void SetUI (UIComponent* _ui) { ui = _ui; }
    ConfigAPI (UIComponent* _ui = NULL) : configData("config"), ui (_ui){}
};
```



Διαλογικά εργαλεία II (12/15)

```
class UIComponent {
protected:
    ConfigData          uiAttrs;      // actual ui display attrs
    ConfigAPI           configAPI;   // turned to a field of the superclass
public:
    const ConfigAPI&   GetConfigAPI (void) const { return configAPI; }
    ConfigAPI&         GetConfigAPI (void) { return configAPI; }
    virtual const string GetId (void) const = 0;

    void               Start (void) {
        // if first time we mirror props from UI to configAPI
        if (configAPI.GetData().Get().empty()) // no props?
            configAPI.GetData().Set(uiAttrs.Get());
        else    // just assign values
            configAPI.GetData().Assign(uiAttrs);
    }

    virtual void        Draw (const ConfigData& data) const = 0;

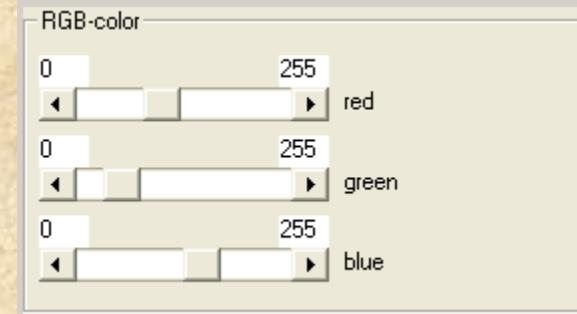
    void               Apply (void) { // write UI component attrs to config
        uiAttrs.Assign(configAPI.GetData());
        Draw(uiAttrs);
    }
    UIComponent (void) : uiAttrs("config"){}
};
```



Διαλογικά εργαλεία II (13/15)

```
class ColorProperty : public AggregateProperty { ←  
    public:  
        ColorProperty (void) : AggregateProperty("RGB-color") {  
            Add(new IntRangeProperty("red", 0, 255, 0));  
            Add(new IntRangeProperty("green", 0, 255, 0));  
            Add(new IntRangeProperty("blue", 0, 255, 0));  
        }  
};  
struct Color { ←  
    unsigned char r, g, b;  
};
```

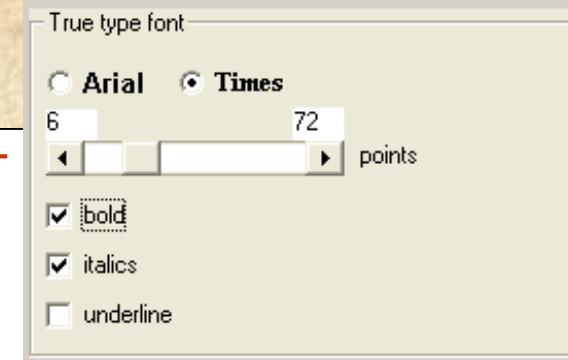
Υποθέτουμε ότι αυτή είναι
η native δομή του GUI library
για colors





Διαλογικά εργαλεία II (14/15)

```
class FontProperty : public AggregateProperty { ←  
public:  
    FontProperty (void) : AggregateProperty("True type font") {  
        vector<string> families;  
        families.resize(2);  
        families[0] = ("Times");  
        families[1] = ("Arial");  
        EnumeratedProperty* family;  
        family = new EnumeratedProperty("Family", families, 0); // default: Times  
        Add(family);  
        Add(new IntRangeProperty("points", 6, 72, 12));  
        Add(new BooleanProperty("bold", false));  
        Add(new BooleanProperty("italics", false));  
        Add(new BooleanProperty("underline", false));  
    }  
};  
struct Font { ←  
    std::string family;  
    unsigned points;  
    bool italics;  
    bool underline;  
};
```



Υποθέτουμε ότι αυτή είναι
ή native δομή του GUI library
για fonts



Διαλογικά εργαλεία (15/15)

```
class Editor : public UIComponent {
public:
    void           Draw (const ConfigData& data) const {
        ColorProperty& color = (ColorProperty&) *data.Get("RGB-color");
        FontProperty& font  = (FontProperty&) *data.Get("True type font");
        // now draw using 'font' and 'color' values
    }
    virtual const string
                  GetId (void) const { return "Editor"; }

    Editor (void) {
        uiAttrs.Add(new FontProperty);
        uiAttrs.Add(new ColorProperty);
        GetConfigAPI().SetUI(this);
        ConfigAPIs::Add(&GetConfigAPI());
    }
    ~Editor (void) {
        GetConfigAPI().SetUI((UIComponent*) 0);
    }
};
```



Ένθετο (1/3)

```
// implementation of Debugger component
class Debugger : UIComponent {
private:
    void FillConfigData (void);
    AggregateProperty* debuggerConfigData = nullptr;
public:
    Debugger(..) {
        debuggerConfigData = &uiAttrs;
        FillConfigData();
    }
};
```

Assume we have
Debugger UI component,
split into a number of
other components

```
❑ // without our design pattern we would have to
  // define a struct like this.
❑ struct debuggerConfigData {
    ...
};
```



Ένθετο (2/3)

```
void Debugger::FillConfigData (void) {  
    FillCallStackConfigData();  
    FillExprEvaluatorConfigData();  
    // invoke all the rest of Fill<Comp>ConfigData();  
    ...  
}
```

In this method we invoke the respective *fill* methods of contained components

```
void FillCallStackConfigData (void) {  
  
    auto* callStack = new AggregateProperty("call stack");  
    debuggerConfigData->Add(callStack);  
  
    // an example for defining a call stack font  
    callStack->Add(  
        new IntRange("font point size", 8, 72, 1)  
    );  
    std::vector<std::string> domain;  
    domain.push_back("Times");  
    domain.push_back("Arial");  
    ...  
    callStack->Add(  
        new EnumeratedProperty("font family", domain, 0)  
    );  
}
```

Sample call stack attributes, normally we expect to add more...



Ένθετο (3/3)

```
// without properties we still have to define C-structs
// (or C++ data classes)

struct callStackConfigData {
    enum FontFamily { Times, Arial, ... };
    FontFamily fontFamily = Times;
    unsigned pointSize = 1; // 8..72
};

struct debuggerConfigData {
    callStackConfigData callStackData;
};
```

The old traditional way of defining attributes: we end-up with an isomorphic matching of the C-structures and the respective aggregate hierarchy.



Αυτόματη παραγωγή διεπαφής (1/8)

■ Αλγόριθμος αυτόματης παραγωγής του configuration interface

Produce a *menu* M with all ids of the configuration APIs in the holder

For each configuration API instance C in the holder Do

 Produce a *dialogue box* D with *variant tab lists* T with *label* the C.GetId()

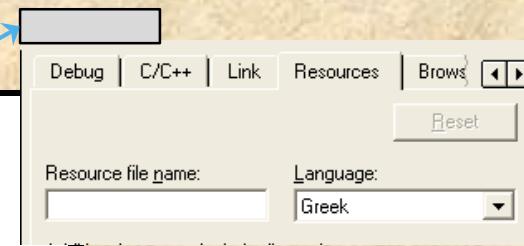
 For each property P in C.properties Do

Begin

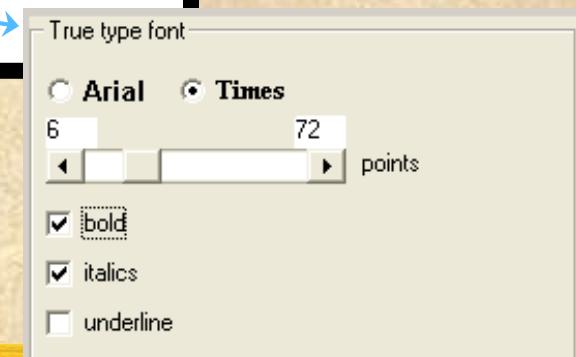
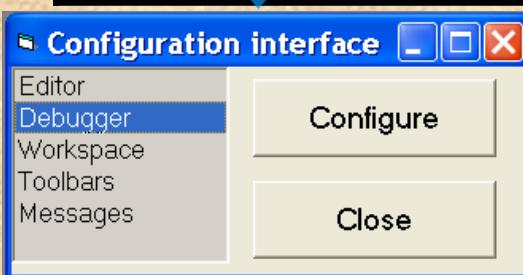
 Add a *new scrollable tab entry* in D with the P.Id

 Make *a micro interface* for P as in this tab entry and add it in T,
 while forcing inner aggregates to be produced as *scroll views* (with
 nesting if needed).

End

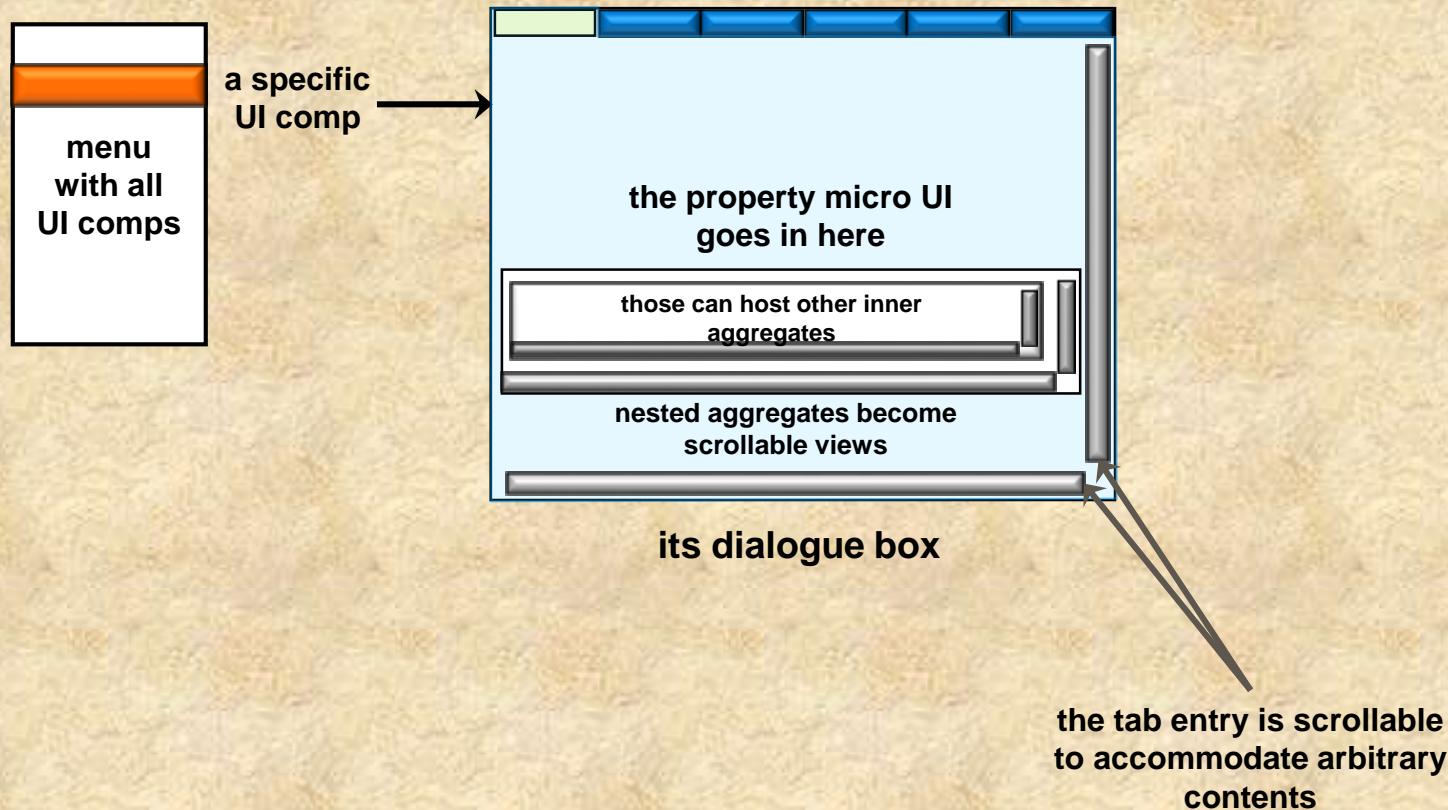


One tab list per UI component, one tab entry per configurable property





Αυτόματη παραγωγή διεπαφής (2/8)





Αυτόματη παραγωγή διεπαφής (3/8)

- Η μέθοδος αυτή ουσιαστικά παράγει αυτόματα μία διεπαφή με αφετηρία ένα δομημένο τύπο δεδομένων
- Η διεπαφή αυτή δίνει τη δυνατότητα να γίνονται edited τιμές αυτού του τύπου
- Η μέθοδος σχετίζεται με μία ευρύτερη οικογένεια συστημάτων που λέγονται User Interface Management Systems
 - ειδικότερα την υποκατηγορία των semantic-based interface generators
 - από ένα API definition παράγουν με ευρεστικό τρόπο μία διεπαφή η οποία προσφέρει αλληλεπιδραστικά το API
 - Φυσικά δεν δουλεύουν για όλες τις περιπτώσεις ικανοποιητικά



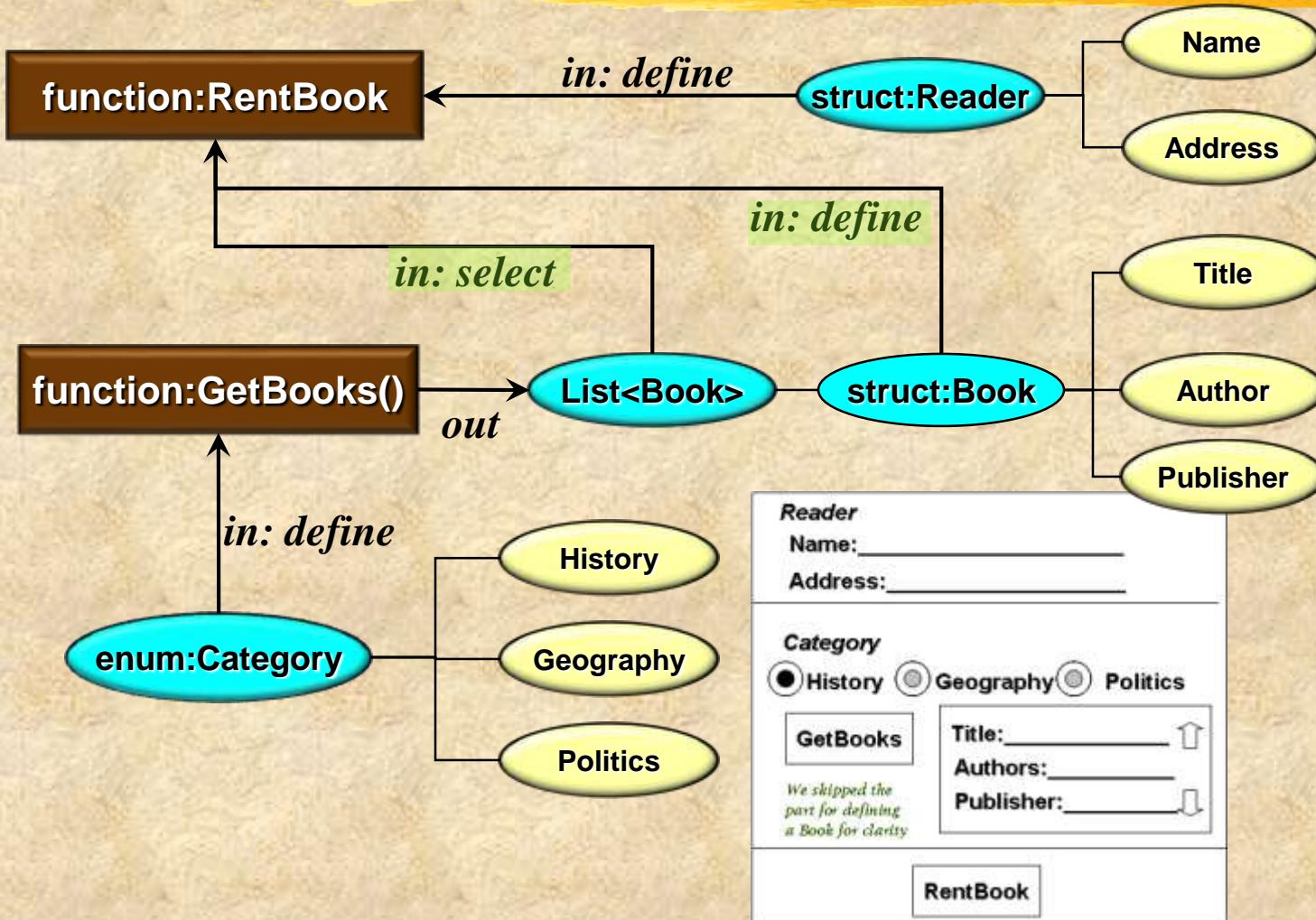
Αυτόματη παραγωγή διεπαφής (4/8)

```
enum Category { History, Geography, Politics};  
struct Book { String title, authors, publisher; };  
function GetBooks (in: Category, out: List<Book>);  
struct Reader { string name, address};  
function Rent (in: Book, in: Reader);
```

- An **enum** type maps to single choice selections
- Every **function** becomes a **push button**
- Function buttons are enabled **only when all arguments are available**
- Every **argument** maps to an editing UI based on its type
 - *They are filled in manually by the user*
- Function **results** map to UIs depending on their type
 - *They are filled in automatically after functions are invoked*
- Argument **selection** UI is produced for arguments matching the return type of some functions
 - *They can be chosen instead of being explicitly edited*



Αυτόματη παραγωγή διεπαφής (5/8)



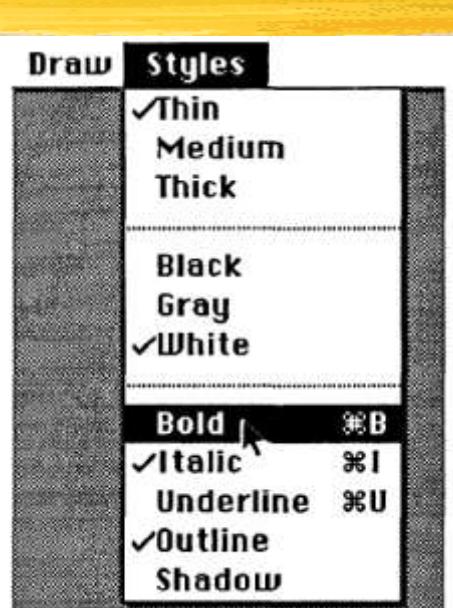
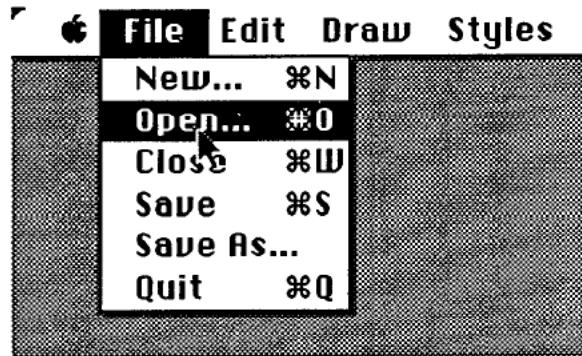


Αυτόματη παραγωγή διεπαφής (6/8)

```

PROCEDURE NewDrawing (
  (* Menu=File Name='New...' Key=N *)
  DrawFile : OutFileDesc);
PROCEDURE OpenDrawing (
  (* Menu=File Name='Open...' Key=O *)
  DrawFile : InFileDesc);
PROCEDURE CloseDrawing;
(* Menu=File Name=Close Key=W *)
PROCEDURE SaveDrawing;
(* Menu=File Name=Save Key=S *)
PROCEDURE SaveDrawingAs (
  (* Menu=File Name='Save As...' *)
  DrawFile : OutFileDesc);

```



TYPE

```

LineWidth = (ThinLine (*Name=Thin *),
  MedLine (*Name=Medium*),
  ThickLine (*Name = Thick *));
FillColor = (BlackFill (*Name=Black*),
  GrayFill(*Name=Gray*),
  WhiteFill (*Name=White*) );

```

VAR

```

LineSize (* Menu=Styles *): LineWidth;
ColorSetting (* Menu=Styles *): FillColor;

```

VAR

```

BoldText (* Menu=Styles
  Name=Bold Key=B*) : Boolean;
ItalicText (* Menu=Styles
  Name=Italic Key=I*) : Boolean;
UnderLineText (* Menu=Styles
  Name=Underline Key=U*) : Boolean;
OutlineText (* Menu=Styles
  Name=Outline *) : Boolean;
ShadowText (* Menu=Styles
  Name=Shadow *) : Boolean;

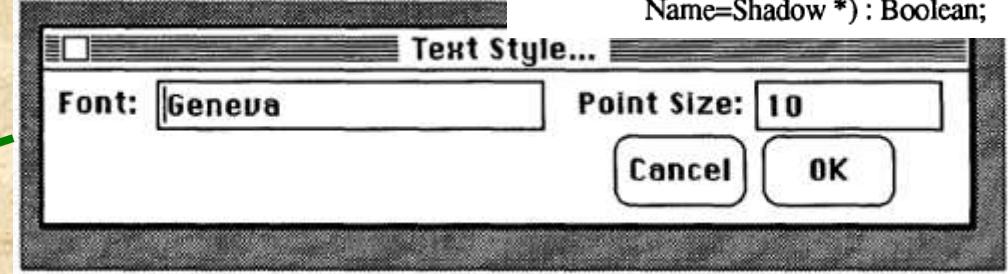
```

```

TYPE Str40 = STRING[40]
TextStyle = RECORD
  Font : Str40;
  Points (*Name='Point Size'*): Integer;
END;

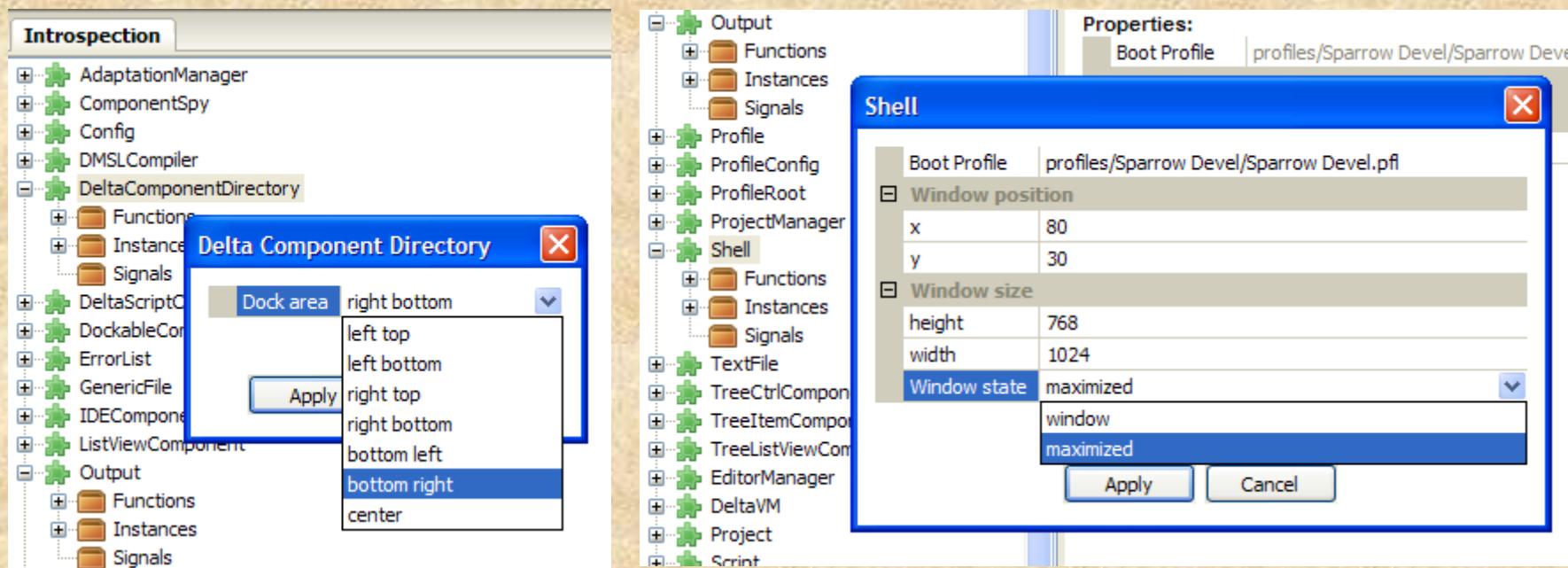
VAR StyleOfText(*Menu=Styles
  Name='Text Style...'): TextStyle;

```





Αυτόματη παραγωγή διεπαφής (7/8)

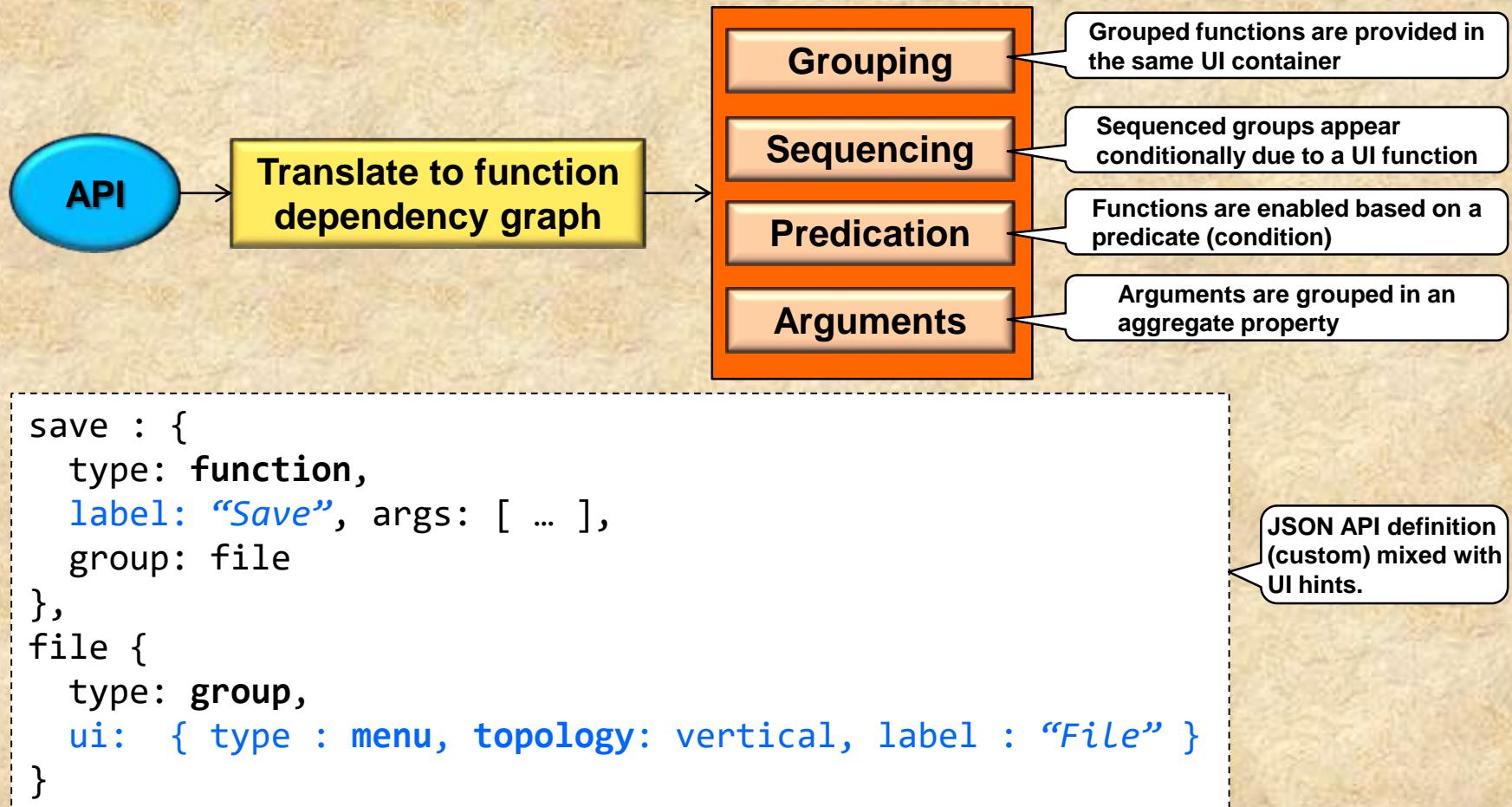


Εφαρμογή της μεθόδου για αυτόματη παραγωγή της διεπαφής για την προσαρμογή των active components μέσω του introspection window στο Sparrow IDE της Delta

<http://www.ics.forth.gr/hci/files/plang/sparrow-setup.exe>



Αυτόματη παραγωγή διεπαφής (7/8)





Περιεχόμενα

- Υλοποίηση διαλογικών εργαλείων προσαρμογής
- ***Ενσωματωμένοι διάλογοι προσαρμογής***
- Προσαρμογές με scripting languages
- Επίλογος



Ενσωματωμένοι διάλογοι προσαρμογής (1/3)

- Στην περίπτωση αυτή, η δυνατότητα προσαρμογής υποστηρίζεται μέσα από το ίδιο το τμήμα της διεπαφής
 - επιτυγχάνεται μεγαλύτερη ταχύτητα σε περίπτωση που ο χρήστης επιθυμεί να εφαρμόσει κάποιες προσαρμογές
 - επιτυγχάνεται καλύτερη ποιότητα διαλόγου, αφού οι προσαρμογές δίνουν επιπλέον ευελιξία
 - ουσιαστικά δεν ακολουθείται το προηγούμενο σχεδιαστικό πρότυπο
 - επειδή υπάρχει ανεξαρτησία ως προς τον τρόπο υλοποίησης των προσαρμογών ανά τμήμα, υπάρχει κίνδυνος επανάληψης παρόμοιου κώδικα, ή λύσης του ίδιου προβλήματος με διαφορετικό τρόπο
 - συνήθως απαιτούνται πολύ λίγες και απλές ενέργειες από τον χρήστη για κάθε προσαρμογή (click, drag, drag & drop).



Ενσωματωμένοι διάλογοι προσαρμογής (2/3)

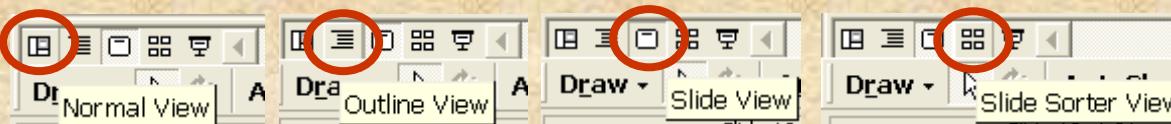
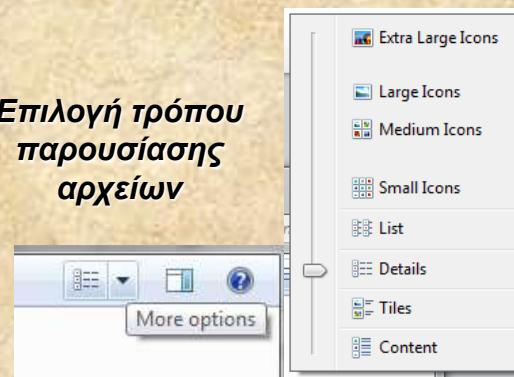
■ Παραδείγματα

Name	Size	Type	Date Modified
Lecture6.ppt	248 KB	Microsoft P...	19/4/2004 2:35 μμ
Lecture7.ppt	343 KB	Microsoft P...	20/4/2004 6:11 μμ

Name	Size	Type	Date Modified
Lecture12....	297 KB	Microsoft P...	6/5/2004 5:20 μμ
Lecture11....	309 KB	Microsoft P...	6/5/2004 5:21 μμ

Ρύθμιση τρόπου παρουσίασης με βάση το όνομα αρχείου

Επιλογή τρόπου παρουσίασης αρχείων



Ρύθμιση της οπτικής διάταξης των toolbars

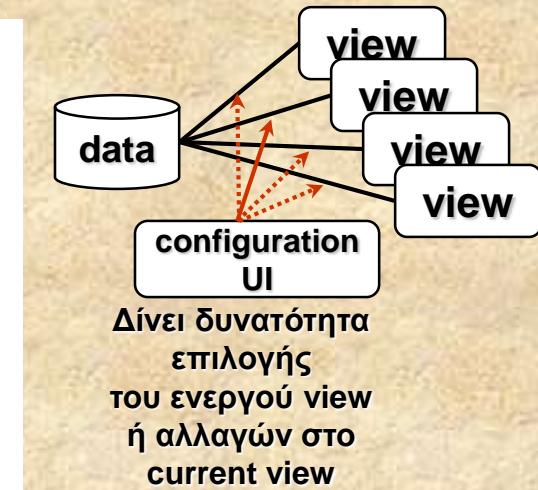
Ρύθμιση των λειτουργιών που εμφανίζονται στα toolbars

Ρύθμιση τρόπου παρουσίασης των slides



Ενσωματωμένοι διάλογοι προσαρμογής (3/3)

- Συνήθως κατασκευαστικά είναι συγγενείς με τις τεχνικές υλοποίησης πολλαπλών αναπαραστάσεων δεδομένων (γνωστό και σαν περίπτωση του View pattern).
- Αν και η δυνατότητα switching μεταξύ διαφορετικών views μπορεί απλά να θεωρηθεί ως μία υποστηριζόμενη λειτουργία, λόγω του ότι δίνει δυνατότητα διαφορετικών επιλογών σε διαφορετικούς χρήστες, χαρακτηρίζεται ως προσαρμογή.
- Προγραμματιστικά δεν έχει καμία συγγένεια με την αυτόματη παραγωγή διεπαφής, παρά μόνο εάν οι διάφορες εναλλακτικές αναπαραστάσεις έχουν χαρακτηριστικά που υφίστανται προσαρμογής. Άλλα στην περίπτωση αυτή απλώς έχουμε συνδυασμό των δύο μεθόδων.





Περιεχόμενα

- Υλοποίηση διαλογικών εργαλείων προσαρμογής
- Ενσωματωμένοι διάλογοι προσαρμογής
- **Προσαρμογές με *scripting languages***
- Επίλογος



Προσαρμογές με scripting languages (1/13)

- Προσφέρεται μία scripting γλώσσα (θα θυμηθούμε σε λίγο τι σημαίνει αυτό), ώστε ο χρήστης, ανάλογα με τις δυνατότητες και γνώσεις που έχει, να μπορεί:
 - να προσαρμόσει την «συμπεριφορά» του συστήματος,
 - και ουσιαστικά να το επεκτείνει (προσθέτοντας νέες λειτουργίες και διάφορους αυτοματισμούς)
- Μόνο συστήματα μεγάλης κλίμακας είναι αυτά που προσφέρουν τέτοιες γλώσσες, όταν έχουν δυνατότητες που δεν μπορούν να χρησιμοποιηθούν και να προσαρμοστούν βέλτιστα μόνο μέσω διαλογικών εργαλείων.



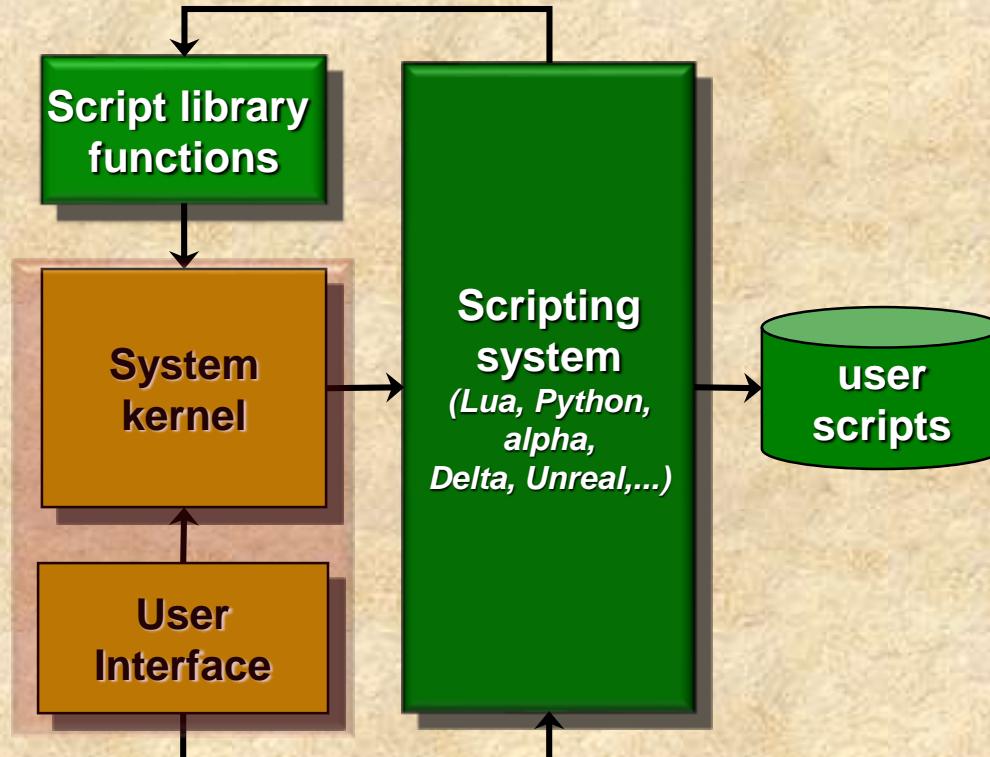
Προσαρμογές με scripting languages (2/13)

- *Scripting languages*. Είναι γλώσσες προγραμματισμού με τα παρακάτω χαρακτηριστικά:
 - προσφέρουν τις βασικές λειτουργίες του κυρίως συστήματος (host system / environment) ως εγγενείς συναρτήσεις (library functions)
 - τρέχουν μόνο μέσα από αυτά τα συστήματα
 - είναι απλές καθώς
 - ◆ δεν έχουν δηλώσεις τύπων (no user-defined types),
 - ◆ οι μεταβλητές δηλώνονται αυτόματα κατά τη χρήση (declaration by use)
 - ◆ οι μεταβλητές παίρνουν δυναμικά τον τύπο της τιμής που τους εκχωρείται, ο οποίος και μπορεί να αλλάζει πολλές φορές (dynamic typing)
 - έχουν συνήθως garbage collection
 - είναι κυρίως interpreted, υπάρχουν όμως και compiled γλώσσες αυτού του τύπου
 - δεν προσφέρεται συνήθως ολοκληρωμένο περιβάλλον ανάπτυξης (δηλ. δεν αναμένουμε να έχουμε source-level debugger, integrated development environment)



Προσαρμογές με scripting languages (3/13)

■ Συνήθης αρχιτεκτονική



- **Χαρακτηριστικά παραδείγματα**
 - Auto CAD / Auto Lisp
 - 3D Studio Max / Max Script
 - OS Shells / Shell Scripts
 - Quake / Quake C
 - Unreal / Unreal Script
- **Πολλές ομοιότητες με scripting γλώσσες που χρησιμοποιούνται για ανάπτυξη συστημάτων**
 - Flash / Action Script
 - Visual Basic / VB Script
 - Web / Java Script



Προσαρμογές με scripting languages (4/13)

- Μία από τις πλέον οικείες περιπτώσεις προσαρμογών με scripting languages σε προγραμματιστές είναι η χρήση shell scripts και resource files (κυρίως για Unix variants)
 - Μεταβλητές του περιβάλλοντος χρήσης (shell environment variables) μπορούν να λαμβάνουν τιμές με σχετικά πολύπλοκες εκφράσεις
 - Πολύπλοκες και επαναλαμβανόμενες λειτουργίες πάνω σε αρχεία περιγράφονται με γενικά scripts ώστε να αποφεύγεται η χειροκίνητη επανάληψη όποτε αυτό απαιτείται (αυτοματισμός)

```
Αυτόματη μετονομασία των .cpp αρχείων σε .cc
for i in *.cpp
do mv $i `echo $i | awk -F. '{ print$1 }' `.cc done
```

- Σε αυτές τις περιπτώσεις η έννοια της προσαρμογής δεν είναι απλώς η αλλαγή χαρακτηριστικών της διεπαφής αλλά:
 - η προσφορά μίας πιο αποτελεσματικής διεπαφής (scripting) για την προσαρμογή ή επέκταση του συστήματος στις δικές μας ανάγκες.

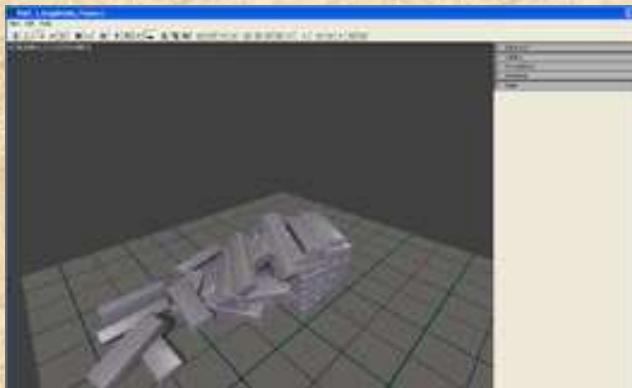


Προσαρμογές με scripting languages (5/13)

- Δημοφιλείς και αρκετά προηγμένες είναι οι scripting languages των video games
 - υποστηρίζουν τα γνωστά *mod extensions*
- Μία τέτοια γλώσσα είναι και η ***Unreal Script*** για τα παιχνίδια της ***Unreal Engine***
 - Για τη σειρά τρισδιάστατων παιχνιδιών δράσης Unreal (όπως Tournament και Death Match)
 - <http://unreal.epicgames.com/UnrealScript.htm>
 - Υβριδική με χαρακτηριστικά OOP, garbage collection, static typing (δηλώνεται ο τύπος των μεταβλητών) και domain specific (έχει keywords και types ειδικά για games)
 - Βασίζεται σε states των αντικειμένων: ο προγραμματιστής υλοποιεί state handlers ενώ μπορεί να αλλάζει το state αντικειμένων
 - Γίνεται compiled σε byte code (δεν είναι ανοικτές οι προδιαγραφές του) και εκτελείται από virtual machine



Προσαρμογές με scripting languages (6/13)



Προσαρμογή των χαρακτηριστικών φυσικής συμπεριφοράς με εξομοίωση κινηματικής

```
state() TriggerToggle {
    function Trigger( actor Other, pawn EventInstigator ) {
        log("Toggle");
        Trigger = Other;
        Direction *= -1;
        Enable( 'Tick' );
    }
}
// Attacking from a distance.
state RangeAttacking expands Attacking {
    // Stick specialized functions here...
}
```



Προσαρμογές με scripting languages (7/13)

- Στην ίδια κατηγορία, αλλά με διαφορετικά χαρακτηριστικά, υφίσταται η **Quake C** για τις εκδόσεις του **Quake**
 - Υποτίθεται ότι μοιάζει στη C. Γίνεται compiled σε byte code ο οποίος εκτελείται από το runtime Quake engine.
 - Απλή στη χρήση, με πολλές συναρτήσεις βιβλιοθήκης της βασικής μηχανής, αλλά μικρή βοήθεια για τη διαδικασία ανάπτυξης
 - ◆ σε σύγκριση με υπάρχουσες γλώσσες γενικού σκοπού ή ακόμη και την ίδια τη γλώσσα C
 - Δεν υποστηρίζει βασικά πράγματα, όπως function call expressions ως ορίσματα σε κλήση συναρτήσεων, λόγω ιδιότυπων επιλογών υλοποίησης
 - ◆ π.χ. το **f(g())** πρέπει να γίνεται **x=g(); f(x)**



Προσαρμογές με scripting languages (8/13)



•Όλα τα αντικείμενα που είναι ενεργά σε μία σκηνή είναι διαθέσιμα σαν entity objects στη γλώσσα. Μπορεί να οριστεί πως αντιδρούν, πως παρουσιάζονται, κλπ, με τη χρήση ειδικών lib functions ανάλογα με τον τύπο τους.

•Ορισμένες «παραξενιές» της γλώσσας οφείλονται στην προσπάθεια να γίνει η όσο το δυνατόν πιο γρήγορη (τώρα είναι 10 φορές περίπου αργότερη από τη C – μεγάλη ταχύτητα για scripting γλώσσα, αν και το ίδιο πετυχαίνει και η Unreal).

```
void () think = {...};  
// C: void think() {...}  
  
entity () FindTarget = {...};  
// C: entity FindTarget() {...}  
  
void(vector destination, float speed, void() callback) SUB_CalcMove = {...};  
// C: void SUB_CalcMove (vector destination, float speed, void (*callback)()) {...}
```



Προσαρμογές με scripting languages (9/13)

- Μία γλώσσα της κατηγορίας scripting για MMORPG, ειδικότερα για το **Second Life**, είναι η **Linden Scripting Language (LSL)**
 - Η δομή της βασίζεται στη Java κα στη C, είναι strongly typed, compiled σε byte code, εκτελείται από virtual machine
 - Ένα script στο Second Life είναι ένα σύνολο εντολών που μπορούν να τοποθετηθούν μέσα σε ένα αρχέτυπο αντικείμενο του κόσμου, αλλά όχι στους χαρακτήρες.
 - ◆ Ωστόσο, οι χαρακτήρες μπορούν να φορούν αντικείμενα με scripts.
 - Τα LSL scripts γράφονται / μεταφράζονται με έναν editor / compiler που είναι ενσωματωμένος στον world editor που προσφέρεται στους παίκτες.
 - Επιτρέπει στα αντικείμενα να αλληλεπιδρούν με τον κόσμο του Second Life και το Internet μέσω email, XML-RPC, και πρόσφατα με HTTP requests



Προσαρμογές με scripting languages (10/13)

```
default
{
    state_entry() ←also an event handler
    {
        llSay(0, "Hello, Avatar!");
    }

    touch_start(integer total_number)
    {
        llSay(0, "Touched.");
    }
}

default ←state-specific code
{
// contents of state go here
}

state playing ←state-specific code
{
// this is a state called "playing"
}

default
{
    ↵event handler
    touch_start(integer total_number)
    {
        llSay(0, "Hello World");
    }
}
```

- Είναι μία state-based και even-based language, δηλαδή βασίζεται σε states των αντικειμένων και event τα οποία λαμβάνουν χώρα κατά τη διάρκεια του παιχνιδιού.
- Π.χ., μια πόρτα μπορεί να είναι *open* or *closed* και ένα φωτιστικό *on* ή *off*. Ένα πρόσωπο μπορεί να είναι *hyperactive*, *calm*, ή *bored*. Η τεχνική βασίζεται στο γεγονός ότι πολλές συμπεριφορές μπορούν να μοντελοποιηθούν με states. Αυτό θα το μελετήσουμε με λεπτομέρεια αργότερα στους τεχνητούς χαρακτήρες.
- Τα είδη των events δεν είναι επεκτάσιμα από το χρήστη στο Second Life, αλλά είναι προκαθορισμένα. Προκαλούνται είτε από αντικείμενα και χαρακτήρες που αλληλεπιδρούν με τον κόσμο ή δημιουργούνται μέσα σε scripts. Προκαλούν την κλήση των event handlers.



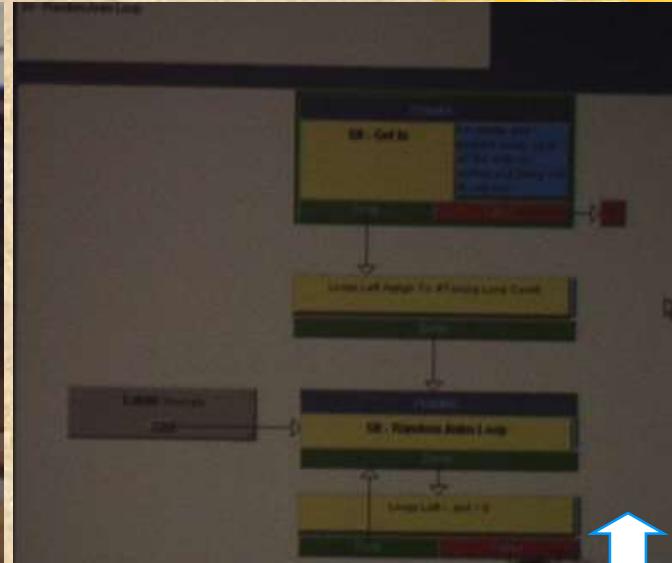
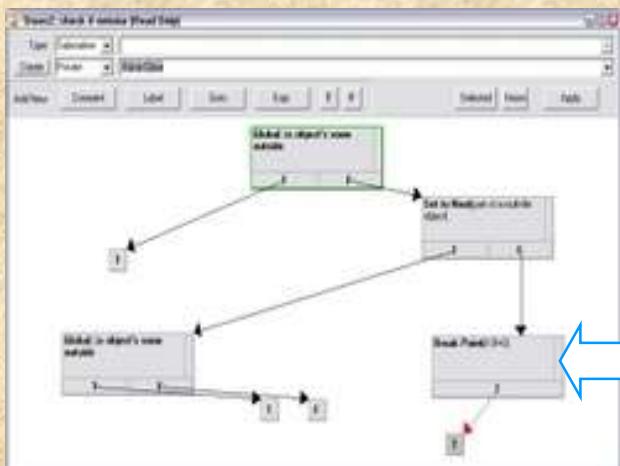
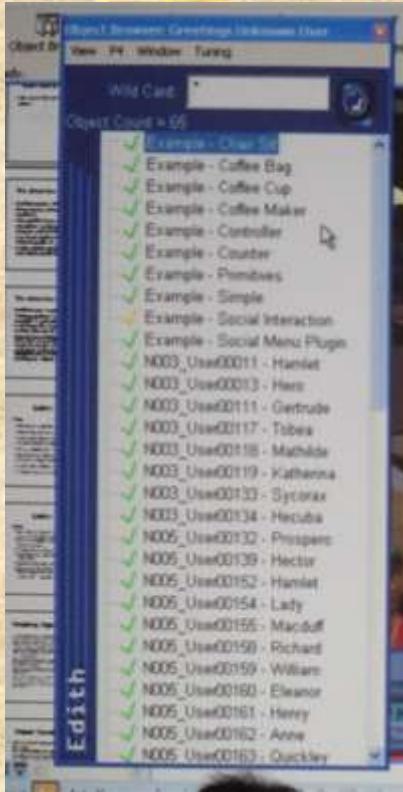


Προσαρμογές με scripting languages (11/13)

- Μία γλώσσα για scripting τεχνητών χαρακτήρων για people simulators είναι η **SimAntics** για την ομάδα παιχνιδιών **The Sims**
 - Είναι interpreted γλώσσα κυρίως για να μπορούν να γίνονται γρήγορα διορθώσεις στα script sources την ώρα που το παιχνίδι εκτελείται
 - ◆ Αυτή η ευκολία είναι γνωστή ως edit-and-continue
 - Κάθε αντικείμενο τρέχει το δικό του interpreter με τη δική του στοίβα, ενώ πριν από κάθε function call γίνεται ένα back-up του object state
 - ◆ Εάν το script κάνει crash, τότε γίνεται restart και το αντικείμενο restored στο backup state
 - Το προηγούμενο υποστηρίζεται με την έννοια των micro-threads (όχι system threads), που σχετίζεται με την έννοια των continuations
 - ◆ lightweight threads με user-defined context-switching points



Προσαρμογές με scripting languages (12/13)



Η λογική των χαρακτήρων και η αλληλεπίδραση μεταξύ των scripts προγραμματίζεται με ένα script tree editor

Τα scripts επειδή είναι κυρίως state transitions μπορούν να προγραμματίζονται με ειδικό γραφικό εργαλείο.



Προσαρμογές με scripting languages (13/13)

- Εάν θέλετε να υποστηρίξετε μία τέτοια τεχνική προσαρμογής, θα πρέπει να κάνετε τα εξής:
 - Σχεδίαση και υλοποίηση της γλώσσας (συνήθως αυτό απαιτεί και τη σχεδίαση του virtual machine)
 - Διάθεση του compiler στους χρήστες
 - Μεταφορά όλων των βασικών λειτουργιών του συστήματός σας ως library functions
 - Υποστήριξη της βασικής λειτουργίας εκτέλεσης εξωτερικού κώδικα (script entry point)
- Πρόκειται για μία από τις πιο προηγμένες μεθόδους προσαρμογής που απαιτεί πολύ μεγάλη τεχνογνωσία για να μπορεί να υποστηριχθεί.
 - Το πιο συνηθισμένο είναι να χρειαστεί να μάθετε καλά μία τέτοια γλώσσα για να «χτίσετε» πάνω από ένα υπάρχον σύστημα



Προσαρμογές με scripting languages – ένθετο



Poll Results			
Which language do you use for scripting in your game engine?			
Python (pure)	█	6.98%	48
Python (stackless)	█	0%	0
Ruby	█	1.16%	8
Perl	█	1.31%	9
C (with co-routines)	█	9.75%	67
Lisp	█	1.45%	10
TCL	█	0.582%	4
Lua	█	20.5%	141
I made my own	█	26.3%	181
My engine doesn't have scripting	█	27.3%	188
Other	█	4.51%	31
[Previous Polls]		Total Votes: 687	

<http://www.gamedev.net/gdpolls>

Poll Results			
What computer programming language do you primarily use?			
C++	██████████	69%	912
Java	█	5.9%	78
C#	███	15.8%	209
Visual BASIC	█	1.43%	19
Python	█	1.89%	25
DarkBASIC	█	0.378%	5
Other	█	5.52%	73
[Previous Polls]		Total Votes: 1321	

→ Uses low-level argument push and extraction to call C functions loaded as DLLs.

→ Lua (1993 -) is a dynamically-typed compiled and interpreted language, built in C, enabling extension by scripting of C programs. The *alpha* language is a large subset of the Lua language.
<http://www.lua.org/>

Poll Results			
What IDE? Dev C++, VC++ or other?			
Dev C++	█	9.16%	94
VC++	██████████	71.4%	733
Other	███	19.3%	199
[Previous Polls]		Total Votes: 1026	



Περιεχόμενα

- Υλοποίηση διαλογικών εργαλείων προσαρμογής
- Ενσωματωμένοι διάλογοι προσαρμογής
- Προσαρμογές με scripting languages
- **Επίλογος**



Επίλογος (1/2)

- Αναλύσαμε τεχνικά τη μία πλευρά της έννοιας «έξυπνη διεπαφή».
- Είναι λίγο «οξύμωρο σχήμα» ο γεγονός ότι η εξυπνάδα αυτού του είδους συνίσταται σε παθητική παροχή εργαλείων και ευκολιών για τον τελικό χρήστη.
- Ωστόσο, το γεγονός ότι δίνεται η δυνατότητα χειροκίνητης προσαρμογής της διεπαφής, με ποικίλους τρόπους, στις ανάγκες του εκάστοτε χρήστη είναι ένδειξη τεχνικής υπεροχής της διεπαφής και του συνολικού συστήματος.
- Άλλωστε υπάρχουν περιπτώσεις στις οποίες με αυτόματο τρόπο είναι αδύνατο το σύστημα να μαντέψει όλες τις επιθυμητές προσαρμογές που θα ήθελε ο τελικός χρήστης.



Επίλογος (2/2)

- Συνεπώς, αν και συζητάμε για «έξυπνα σχεδιασμένες ευέλικτες διεπαφές», θεωρούμε ότι η υλοποίηση τέτοιων μηχανισμών τις καθιστά, έστω και οριακά, «τεχνητώς ευφυείς».
- Ακόμη και εάν μία διεπαφή προσφέρει αυτόματες προσαρμογές (αυτές που θα μελετήσουμε αργότερα) ποτέ δεν μπορεί να θεωρηθεί ότι η υποστήριξη χειροκίνητων προσαρμογών είναι πλεονάζουσα και άχρηστη
- Ιδιαίτερα όταν οι τεχνικές που παρέχονται είναι πολύ προηγμένες, όπως scripting languages, γεγονός που καθιστά την αυτόματη δημιουργία προσαρμογών ως ένα πολύ δύσκολο πρόβλημα
 - πχ, αν και επιθυμητό, είναι μάλλον πολύ ακριβή η δημιουργία συστήματος που θα παρήγαγε αυτόματα shell scripts παρατηρώντας και αυτοματοποιώντας τις συνήθειες χρήσης