



Multi-Dimensional Context-Aware Adaptation of Service Front-Ends

Project no. FP7 – ICT – 258030

Deliverable D.6.2.2 Standardization Actions Report (Upd. 2)



Due date of deliverable: 30/09/2013

Actual submission to EC date: 30/09/2013

Project co-funded by the European Commission within the Seventh Framework Programme (2007-2013)

Dissemination level

PU

Pubic

Yes

For updates, see: http://www.serenoa-fp7.eu/wp-content/uploads/2013/09/SERENOA_D6.2.2.pdf

Table of Contents

1 Introduction	4
2 Potential opportunities for standardization	4
2.1 Task Models	4
2.2 Domain Models	5
2.3 Abstract UI Models	5
2.4 Concrete UI Models	6
2.4.1 WIMP (desktop GUI)	7
2.4.2 Touch-based GUI (smart phones and tablets)	8
2.4.3 Vocal UI	8
2.4.4 Multimodal UI	10
2.4.5 Industrial UI	10
2.5 Context of Use	11
2.5.1 General Considerations	11
2.5.2 Industry: Fulfilment of Safety Guidelines	12
2.5.3 Automotive: Mitigation of Driver Distraction	12
2.6 Multidimensional Adaptation of Service Front Ends	13
2.6.1 CARF Reference Framework	13
2.6.2 CADs Design Space	14
2.6.3 CARFO Multidimensional Adaptation Ontology	14
2.7 Design-time adaptation rules	15
2.8 Run-time adaptation rules	16
2.9 Advanced Adaptation Logic Description Language (AAL-DL)	17
2.10 Are Event-Condition-Action Rules Needed?	19
2.11 Abduction and Constraint Satisfaction for Model-Based Design	20
2.11.1 How do user interfaces vary across different devices?	22
2.12 Corporate Rules for Consistent User Experience	24
3 D6.2.2 Prospective Standards Development Organizations	24
3.1 W3C Organization and Process	25
4 W3C Model-Based UI Working Group	26
4.1 Introduction	26
4.2 History	27
4.2.1 MBUI Incubator Group	28
4.2.2 MBUI Workshop	29
4.2.3 Formation of MBUI Working Group	30
4.3 MBUI Working Group Charter	32
4.4 MBUI Submissions	34
4.4.1 Advanced Service Front-End Description Language (ASFE-DL)	34
4.4.2 The ConcurTaskTrees Notation (CTT)	35
4.4.3 Useware Markup Language (UseML)	38
4.4.4 User Interface Markup Language (UIML)	40
4.4.5 Abstract Interactor Model (AIM) Specification	41
4.4.6 Multimodal Interactor Mapping (MIM) Model Specification	43
4.4.7 UsiXML	43

4.4.8 MARIA	45
4.5 MBUI WG Note - Introduction to Model-Based UI Design	50
4.6 MBUI WG Note - Rental Car Use Case	50
4.7 MBUI WG Note - Glossary of Terms	52
4.8 MBUI WG Specification - Task Models for Model-Based UI Design	52
4.9 MBUI WG Specification - Abstract User Interface Models	54
4.10 Relationship to IndieUI Working Group	54
4.11 Future Plans	55
5 W3C Ubiquitous Application Design Community Group	56
6 CoDeMoDIS proposal for a COST Action.	58
7 ISO 24744 standardisation action.	59
8 Conclusions	59
9 References	60

1 Introduction

This report describes standardization actions for the Serenoa project, and will consider opportunities for standardization, current progress, and future plans. Our motivation for work on standardization is to encourage the development and uptake of interoperable tools at both design and run-time for context aware model-based user interfaces.

This is the final report on standardization actions for the Serenoa project. Since the last report, we have made progress on the work items in the W3C Model-Based User Interfaces Working Group, and have launched a W3C Community Group to foster dialogue between developers and researchers with a view to future standardization.

The following diagram illustrates the Serenoa Architecture, and many of the components shown will be considered in later sections of this report from the perspective of their potential for standardization.

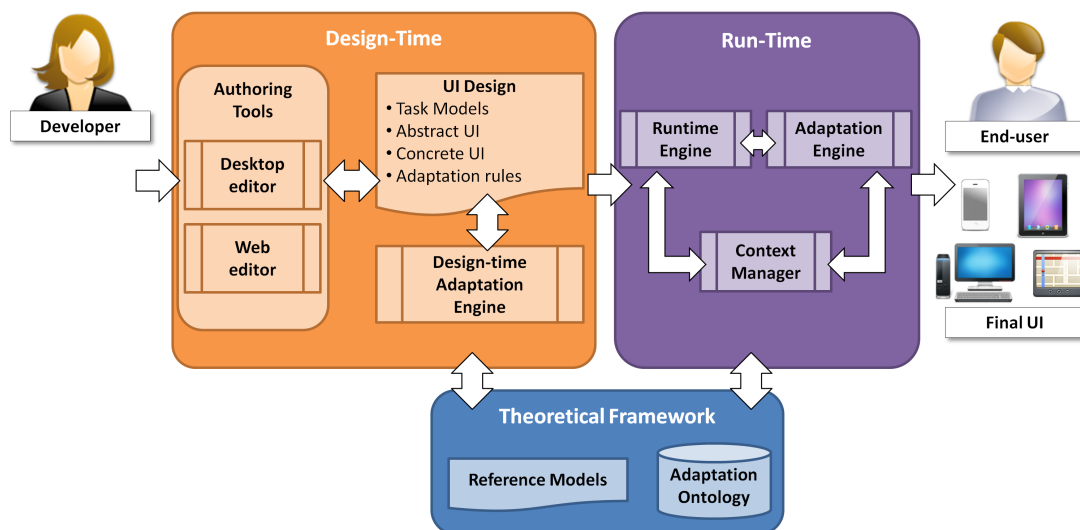


Figure 1: Serenoa Architecture

For an introduction to the architecture and the benefits for a range of stakeholders, you are invited to read the [Serenoa White Paper](#) [SerenoaWP].

2 Potential opportunities for standardization

This section reviews the different areas of work underway in the Serenoa project and provides a brief account of their potential for standardization.

2.1 Task Models

Task models provide a means for describing the set of tasks involved in an interactive system, how the tasks decompose into subtasks, which tasks are to be carried out by

the user, the system or both, and the temporal sequence and inter-dependencies of tasks. Task models enable the user interaction to be described and reviewed without being distracted by the details of the user interface. As such task models are not intended to be a complete description.

The primary task modeling language in Serenoa is ConcurTaskTrees [CTT]. This has good prospects for standardization and would enable interoperable exchange of task models between different user interface design tools. See section 4 on W3C MBUI Working Group for information on how this is proceeding.

2.2 Domain Models

The general architecture for Serenoa assumes a clean separation between the user interface and the application back-end. The interface is defined through a domain model with named properties and methods. Each property can have an atomic value such as a boolean, a number or a string. Alternatively, a property can have a structured value with subsidiary properties and methods. Property values, method arguments and return values are described with a type language. The domain model may also include a means for the system to signal events or exceptions, for example, an asynchronous change in the context of use, or an error in the user's input. A further consideration is whether a method is synchronous or asynchronous, i.e. it takes sufficient time to execute to have a noticeable impact on the user experience.

Serenoa has so far avoided defining a separate formal language for domain models, and instead has embedded a limited treatment as part of the abstract user interface (ASFE-DL, see [D3.2.3]). An adequate formalization of domain models will be essential for interoperable interchange of user interface designs. The precise requirements will depend on the kinds of interactive systems that are being targeted.

2.3 Abstract UI Models

In the Serenoa architecture, abstract user interface design models describe interactive systems at a greater level of detail than is commonly the case for task models, but are still independent of the target platforms and modes of interaction. The [ASFE-DL] language can be loosely described as follows:

At the top level, the abstract user interface can be described in terms of a set of inter-related dialogues. Each dialogue has a set of interactors which can be thought of as abstract versions of user interface controls. Each interactor is bound to the domain model as well as a variety of properties.

There is a lot of potential for standardizing an abstract user interface design language. However, there are many more such languages than is the case for task models. This will make it harder to standardize due to the need to forge bridges between different camps, through the establishment of common use cases, a shared vocabulary and a synthesis of ideas. As such, ASFE-DL will be just one input into the standardization process.

The list of existing alternatives for AUIs is quite lengthy [Souchon2003]. Next we will provide more detailed information regarding the two AUI languages that comprise the consortium's portfolio of authored and co-authored languages in this field, namely: UsiXML and MARIA.

The *User Interface EXtensible Markup Language (UsiXML)* [Vanderdonckt2004] is an XML-compliant mark-up language to describe user interfaces for multiple contexts and different modalities. UsiXML allows also non-developers to use the language to describe user interfaces, mainly because the elements of the UI can be described at a high level, regardless of the platform of use. The UsiXML language was submitted for a standardisation action plan in the context of the Similar network of excellence and of the Open Interface European project.

MARIA (Model-based language for Interactive Applications) [Paterno2009], is a universal, declarative, multiple abstraction-level, XML-based language for modelling interactive applications in ubiquitous environments. For designers of multi-device user interfaces, one advantage of using a multi-layer description for specifying UIs is that they do not have to learn all the details of the many possible implementation languages supported by the various devices, but they can reason in abstract terms without being tied to a particular UI modality or, even worse, implementation language. In this way, they can better focus on the semantics of the interaction, namely what the intended goal of the interaction is, regardless of the details and specificities of the particular environment considered.

2.4 Concrete UI Models

The concrete user interface involves a commitment to a class of device and modes of interaction. Some typical examples are examined in the following subsections. There are quite a few existing user interface languages at this level of abstraction. Some of these are widely deployed proprietary solutions, where the vendor may feel little imperative to add support for interoperable interchange of user interface designs. An open standard is likely to have a tough time in widening its support beyond a relatively small community of early adopters. The larger the community, the easier it is to gather the resources needed to create and maintain effective easy to use tools and documentation. This is true for both open source and proprietary solutions.

Some examples of existing concrete user interface languages:

- UIML - early example of a user interface markup language
- MXML - introduced by Macromedia for compilation into Flash SWF
- XUL - introduced by Mozilla Foundation for the Gecko engine
- XAML - introduced by Microsoft for use with their .NET framework
- OpenLazlo (LZX) - introduced by Lazlo Systems for their presentation server
- MARIA - developed by ISTI-CNR, and combining abstract and concrete UI
- XForms - developed by W3C for rich forms interfaces

See References section for links to further information.

2.4.1 WIMP (desktop GUI)

The abbreviation WIMP stands for "windows, icons, menus, pointer", and describes the kind of graphical user interface common on desktop computers running operating systems such as Microsoft Windows, MacOS, and Linux + X Windows. WIMP user interfaces were originally developed by Xerox in the early seventies, but came to popular attention through the Apple Macintosh in the mid-eighties, and later Microsoft Windows. A concrete user interface modelling language for WIMP platforms can build upon a wealth of experience. Some examples of common features include:

- scroll-able windows, inline and pop-up dialogues
- click, double click, drag and drop idioms
- window minimization, maximization and close buttons
- icons for minimized applications, and as clickable buttons
- tab controls for groups of related panes
- control bars with subsidiary controls
- drop down menus and combo boxes
- Keyboard short cuts as alternatives to using the mouse/trackpad
- single and multi-line text boxes
- captioned radio buttons
- captioned check boxes
- up/down spinners
- buttons with text and icons as captions
- named boxes for grouping related controls
- a variety of layout policies, e.g. absolute, horizontal, vertical, grid and table layouts

Graphical editors for creating WIMP user interfaces typically consist of a palette of controls that can be dragged on to a canvas. Once there, each control has a set of associated properties that you can update through a property sheet. These can be used to attach the desired behaviour, and it is common to define this with a scripting language that bridges the user interface controls and the application back-end.

One challenge for WIMP user interfaces is adapting to varying window sizes and resolutions. To some extent this can be addressed through layout policies that make the best use of the available space. The end user may be able to vary the font size. Scrollable windows make it possible to view a large window in a smaller screen area. However, large changes in window size and resolution call for more drastic adaptations, and one way to address this via splitting the user interface design into multiple concrete user interface models aimed at different sizes of window.

Some common WIMP UI editing tools and languages:

- Balsamiq
- JustinMind
- Maqetta
- SketchFlow

See References section for links to further information.

2.4.2 Touch-based GUI (smart phones and tablets)

In the last few years, there has been a rapid deployment of phones and tablets featuring a high resolution colour screen with a multi-touch sensor. Touch-based devices typically lack traditional keyboards, and have given rise to a new set of user interface design patterns. Some common features include:

- tap, double tap, long tap, drag and drop
- two finger pinch, stretch and zoom
- swipe to pan
- single rather than multiple windows
- background services
- pop-up notifications
- icons for launching applications
- suspend and resume semantics for applications
- orientation sensing and portrait/landscape adaptation
- ambient light level sensing
- proximity sensing
- GPS-based location sensing
- wide variety of display resolutions
- Bluetooth, USB and NFC interfaces
- variations in support for Web standards, especially scripting APIs

Further study is needed to see just how practical it is to define and standardize a common concrete user interface language for different touch-based platforms such as Apple's iOS and Google's Android. Variations across devices create significant challenges for developers, although some of this can be hidden through the use of libraries.

2.4.3 Vocal UI

Vocal user interfaces are commonly used by automated call centres to provide service that customers can access by phone using their voice and the phone's key pad. Vocal interfaces have to be designed to cope with errors in speech recognition, and ungrammatical or out of domain responses by users. Simple vocal interfaces direct the user to respond in narrow and predictable ways that can be characterized by a speech grammar. Errors can be handled via repeating or rephrasing the prompt, or by giving users the choice of using the key pad. Some relevant existing W3C specifications are:

- Voice Extensible Markup Language (VoiceXML)
- Speech Recognition Grammar Specification (SRGS)
- Semantic Interpretation for Speech Recognition (SISR)
- Speech Synthesis Mark Language (SSML)
- Pronunciation Lexicon Specification (PLS)
- Emotion Markup Language (EmotionML)
- Voice Browser Call Control (CCXML)

- State Chart XML (SCXML)

VoiceXML is similar in some respects to the Hypertext Markup Language (HTML) in its use of links and forms. VoiceXML also provides support for spoken dialogues in terms of error handling, and the use of complementary languages such as SRGS for speech grammars, and SSML for control of speech synthesis and prerecorded speech.

The Serenoa framework can be applied to vocal interfaces described in VoiceXML where the the speech grammars can be readily derived. This is the case for applications involving navigation through a tree of menus, where the user is directed to repeat one of the choices given in a prompt, or to tap the key pad with the number of the choice, e.g.:

```
M: Do you want news, sports, or weather?
U: weather
M: the weather today will be cold and windy.
```

VoiceXML corresponds to the final user interface layer in the Cameleon Reference Framework, and could be complemented by a higher level concrete user interface models for vocal interfaces. Further work is needed to clarify the requirements before standardization can take place.

More sophisticated voice interfaces encourage users to answer in an open ended way, where a statistical language model is used to classify the user's utterance based upon an analysis of large numbers of recorded calls. The classification triggers a state transition network encoding the dialogue model. The following example is from "How may I help you" by Gorin, Parker, Sachs and Wilpon, Proc. of IVITA, October 1996.

```
M: How may I help you?
U: Can you tell me how much it is to Tokyo?
M: You want to know the cost of a call?
U: Yes, that's right.
M: Please hold for rate information
```

This kind of vocal interface is a poor fit for the Serenoa framework as it requires specialized tools for annotating and analyzing large numbers of calls (the above paper cited the use of a corpus of over 10,000 calls), and for the development of utterance classification hierarchies and state transition dialogue models.

State Chart extensible Markup Language (SCXML)

- <http://www.w3.org/TR/scxml/>

SCXML provides a means to describe state transition models of behaviour and can be applied to vocal and multimodal user interfaces.

2.4.4 Multimodal UI

Multimodal user interfaces allow users to provide input with multiple modes, e.g. typing or speaking. A single utterance can involve multiple modes, e.g. saying "tell me more about this one" while tapping at a point on the screen. Likewise the system can respond with multiple modes of output, e.g. visual, aural and tactile, using the screen to present something, playing recorded or synthetic speech, and vibrating the device.

The wide range of possible approaches to multimodal user interfaces has hindered the development of standards. Some work that has been considered includes:

- Using spoken requests to play video or music tracks based upon the Voice Extensible Markup Language (VoiceXML)
- Loosely coupling vocal and graphical user interfaces, where these are respectively described with VoiceXML and HTML, see: <http://www.w3.org/TR/mmi-arch/>
- Extending HTML with JavaScript APIs for vocal input and output, see: <http://www.w3.org/2005/Incubator/htmlspeech/XGR-htmlspeech-20111206/>

The W3C Multimodal Interaction Working Group has worked on

- The Extensible Multimodal Annotation Markup Language (EMMA), which defines a markup language for containing and annotating the interpretation of user input, e.g. speech and deictic gestures.
- Ink Markup Language (InkML), which defines a markup language for capturing traces made by a stylus or finger on a touch sensitive surface. This opens the way to user interfaces where the user writes rather than types or speaks the information to be input.

Human face to face communication is richly multimodal with facial gestures and body language that complements what is said. Some multimodal interfaces try to replicate this for system output by combining speech with an animated avatar (a talking head). See [D5.2.3] for an e-Health example from Telefonica. Handwriting and speech also lend themselves to biometric techniques for user authentication, perhaps in combination of face recognition using video input.

Serenoa could address a limited class of multimodal user interfaces, but it is unclear that it is timely to take this to standardization. A possible exception is for automotive applications where multimodal interaction can be used to mitigate concerns over driver distraction, where drivers need to keep focused on the task of driving safely. W3C's Automotive and Web Platform Business Group is target for dissemination of Serenoa work on adaptive multimodal user interface design.

2.4.5 Industrial UI

There is plenty of potential for applying the Serenoa framework to industrial settings. Manufacturing processes frequently involve complex user interfaces for monitoring

and control purposes. This can combine mechanically operated valves and sensors, together with sophisticated computer based interactive displays. Model-based user interface design techniques could be applied to reduce the cost for designing and updating industrial user interfaces. This suggests the need for work on concrete user interface modelling languages that reflect the kinds of sensors and actuators needed on the factory floor. The need for specialized models for context awareness of interactive systems in industrial settings is covered in a later section.

2.5 Context of Use

This section looks at the context of use and its role in supporting adaptation, starting with general considerations, and then taking a look at industrial and automotive settings.

2.5.1 General Considerations

What is the context of use and how does it assist in enabling context aware interactive systems? There are three main aspects:

1. the capabilities of the device hosting the user interface
2. the user's preferences and capabilities
3. the environment in which the interaction is taking place

Some device capabilities are static, e.g. the size and resolution of the screen, but others change dynamically, e.g. the orientation of the screen as portrait or landscape. Designers need to be able to target a range of devices as people are increasingly expecting to access applications on different devices: a high resolution desktop computer with a mouse pointer, a smart phone, a tablet, a TV or even a car. Model-based techniques can help by separating out different levels of concerns, but this is dependent on understanding the context of use.

We are all individuals, and it is natural for us to expect that interactive systems can adapt to our preferences, and crucially to our own limitations, for instance, colour blindness, a need for increased contrast and for big fonts to cope with limited vision, aural interfaces when we can't see (or have our eyes busy with other matters). Some of us have limited dexterity, and have difficulty with operating a mouse pointer or touch screen. Bigger controls are needed along with the possibility of using assistive technology.

A further consideration is enabling applications to adapt to our emotional state, based upon the means to detect emotional cues from speech. In the car, researchers are using gaze tracking to see what we are looking at, and assessing how tired we are from the frequency of which we blink, as well as the smoothness by which we are operating the car.

Finally, we are influenced by the environment in which we are using interactive systems. Hot/cold, quiet/noisy, brightly lit/dark, the level of distractions, and so forth.

Other factors include the battery level in mobile device, and the robustness or lack of the connection to the network.

From a standardization perspective, there is an opportunity to formalize the conceptual models for the context of use, and how these are exposed through application programming interfaces (APIs) and as properties in the conditions of adaptation rules.

2.5.2 Industry: Fulfilment of Safety Guidelines

Interactive systems for industrial settings need to adapt to dynamic changes in the context of use. A robot arm may need to be kept stationary to allow a human to safely interact with the system. The application thus needs to be able to alter its behaviour based upon sensing the proximity of the user. Another case is where the user must be on hand to monitor the situation and take control of potentially dangerous processes. This suggests the need for specialized models for the context of use in industrial settings.

2.5.3 Automotive: Mitigation of Driver Distraction

Interactive systems in the car pose interesting challenges in the need to keep the driver safely focused on the road, and the risk of legal liability is that isn't handled effectively.

Modern cars have increasingly sophisticated sensors and external sources of information. Some examples include:

- imminent collision detection and braking control
- dynamic adjustment of road-handling to match current conditions, e.g. when there is ice or water on the road
- detection of when the car is veering out of the lane
- automatic dipping of headlights in the face of oncoming traffic
- automatic sensing of road signs
- adaptation for night-time operation
- car to car exchanges of information on upcoming hazards
- access to the current location via GPS
- access to live traffic data over mobile networks
- dead-spot cameras for easier reversing
- sophisticated sensors in many of the car's internal systems

Drivers need to be kept aware of the situation, and free of distractions that could increase the risk of an accident. Phone conversations and entertainment services need to be suspended when appropriate, e.g. when approaching a junction, or the car ahead is slowing down. Safety related alerts need to be clearly recognizable under all conditions. Visual alerts may be ineffective at night due the lights of oncoming traffic, or in the day, when the sun is low on the horizon. Likewise aural alerts may be ineffective when driving with the windows down, or when the passengers are talking noisily.

Automotive represents a good proving ground for the Serenoa ideas for context adaptation. W3C plans to hold a Web and Automotive workshop in late 2012, and to launch standards work thereafter. This provides an opportunity for standardizing models for the context of use, including models of cognitive load, as well as an automotive oriented version of AAL-DL.

2.6 Multidimensional Adaptation of Service Front Ends

The theoretical framework for Serenoa is structured in three components:

- Context-aware Reference Framework (CARF [D2.1.2])
- Context-aware Design Space (CADS [D2.1.2])
- Context-aware Reference Ontology (CARFO [D2.3.2])

Together these provide the concepts and the means for defining, implementing and evaluating context aware interactive systems.

2.6.1 CARF Reference Framework

The Context-aware Reference Framework (CARF) provides core concepts for defining and implementing adaptive and adaptable systems.



Figure 2: Context-aware Reference Framework (CARF)

The above figure illustrates the main axes:

- **What** kinds of things are being adapted, e.g. the navigational flow, or the size of text and images, ...
- **Who** is triggering and controlling the adaption process, e.g. the end user, the system or a third party
- **When** the adaptation takes place, e.g. design-time or run-time.
- **Where** adaptation takes place, e.g. in the device hosting the user interface, in the cloud or at some proxy entity
- **Which** aspects of the context are involved in the adaptation
- **How** is the adaptation performed, i.e. what strategies and tactics are involved

It is unclear how CARF could be standardized. An informative description is fine, but the question to be answered is how CARF is exposed in design tools and at during the run-time of interactive systems.

2.6.2 CADS Design Space

The Context-aware Design Space (CADS) provides a means to analyse, evaluate and compare multiple applications in regards to their coverage level of adaptation, e.g. for dimensions such as modality types.

CADS defines a number of axes for considering adaptation. All of these axes form an ordered dimension, however their levels not always have equal proportions. These are illustrated in figure 3:

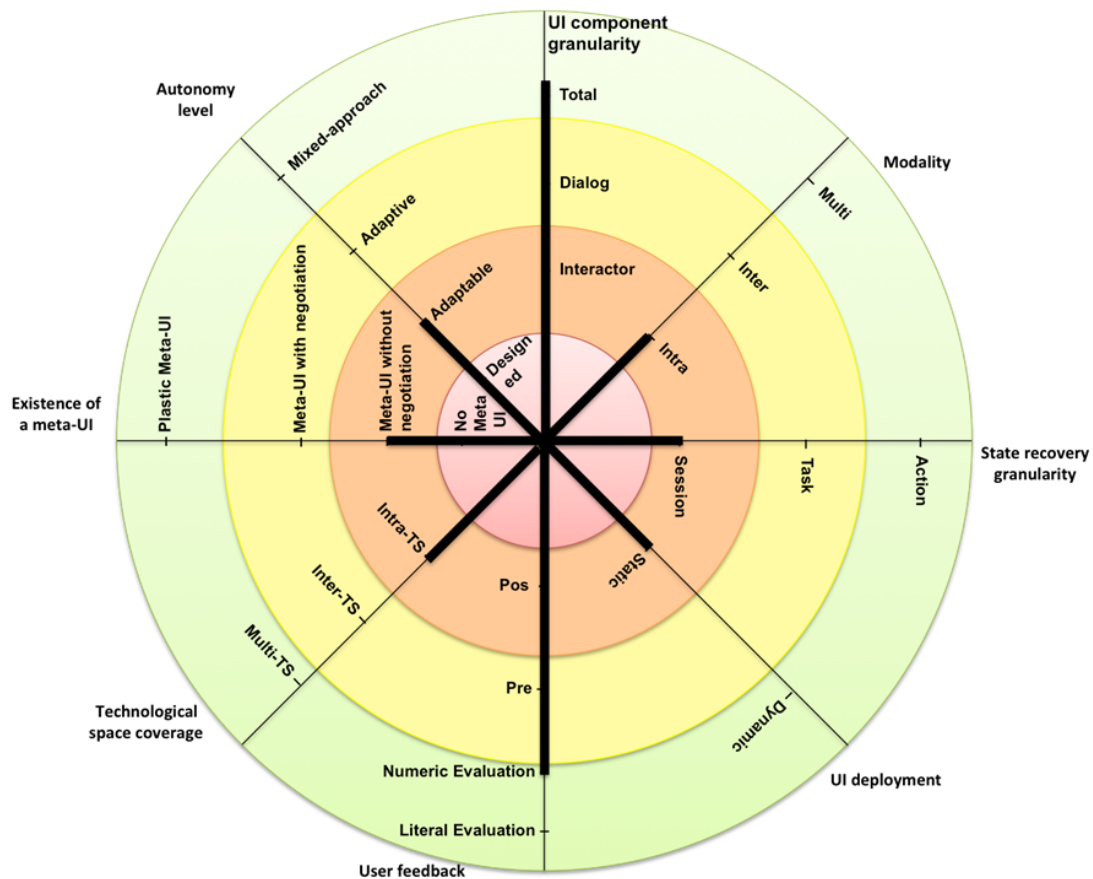


Figure 3: The Context-aware Design Space (CADS)

Designers can use CADS as a conceptual model to guide their thinking. It can also provide a means for classifying collections of adaptation rules. It is unclear at this point just how CADS would feed into standardization, except as a shared vocabulary for talking about specific techniques.

2.6.3 CARFO Multidimensional Adaptation Ontology

The Context-aware Reference Ontology (CARFO) formalizes the concepts and relationships expressed in the Context-aware Reference Framework (CARF). CARFO enables browsing and search for information relevant to defining and implementing the adaptation process. This is useful throughout all of the phases of an interactive system: design, specification, implementation and evaluation.

Standardizing CARFO is essentially a matter of building a broad consensus around the concepts and relationships expressed in the ontology. This can be useful in ensuring a common vocabulary, even if the ontology isn't used directly in the authoring and run-time components of interactive systems.

2.7 Design-time adaptation rules

Design-time adaptation rules have two main roles:

1. To propagate the effects of changes across layers in the Cameleon reference framework [CAMELEON].
2. To provide a check on whether a user interface design complies to guidelines, e.g. corporate standards aimed at ensuring consistency across user interfaces.

One way to represent adaptation rules is as follows:

IF condition THEN conclusion

When executed in a forward chaining mode, rules are found that match the current state of a model, and the conclusion is fired to update the model. This process continues until all applicable rules have been fired. If more than one rule applies at a given instance, a choice has to be made, e.g. execute the first matching rule, or use a rule weighting scheme to pick a rule. Some rule engines permit a mix of forward and backward (goal-driven) execution, where rules are picked based upon their conclusions, and the rule engine then tries to find which further rules would match the conditions.

Forward chaining production rules can be efficiently executed by trading off memory against speed, e.g. using variants of the [RETE] algorithm. Rule conditions can involve externally defined functions, provided these are free of side-effects. This provides for flexibility in defining rule conditions. Likewise, the rule conclusions can invoke external actions. These can be invoked as a rule is fired, or later when all of the applicable rules have fired.

To enable rules to respond to changes in models, the rules can be cast in the form of event-condition-action, where an event corresponds to a change the user has made to the model. Manual changes to the abstract user interface can be propagated to each of the targets for the concrete user interface, for instance desktop, smart phone and tablet. Likewise, manual changes to the concrete user interface for a smart phone can be propagated up to the abstract user interface and down to other targets at the concrete user interface layer.

The set of rules act as an cooperative assistant that applies best practices to help the designer. Sometimes additional information and human judgement is required. The rules can be written to pass off tasks to the human designer via a *design agenda*.

One challenge is to ensure that the maintainability of the set of rules as the number of rules increases. This requires careful attention to separation of different levels of

detail, so that high level rules avoid dealing with details that are better treated with lower level rules.

The above has focused on IF-THEN (production rules) that can respond to incremental changes in models. An alternative approach is to focus on transformation rules that map complete models from the abstract user interface to models for the concrete user interface. W3C's [XSLT] language provides a great deal of flexibility, but at the cost of transparency maintainability. Other work has focused on constrained transformation languages, e.g. the Object Management Group's [QVT] (Query/View/Transformation) languages for transforming models.

There is an opportunity to standardize a rule language for design-time use. When bringing this to W3C, it will be important to show how the rule language relates to W3C's generic Rule Interchange Framework [RIF].

Note that the Serenoa Advanced Adaptation Logic Description Language [AAL-DL] is covered in a subsequent section.

2.8 Run-time adaptation rules

Run-time rules are designed to describe how the user interface should adapt to changes in the context of use. This could be to match the user's preferences or capabilities, or to a change in the environment. The event-condition-action pattern is well suited for this purpose, where events are changes in the context of use, or in the user interface state. Serenoa is exploring this approach with the Advanced Adaptation Logic Description Language [AAL-DL].

The examples considered so far have focused on high level adaptations with the idea of invoking separate adaptation modules to determine the detailed changes that need to be applied. These modules could be implemented with production rules, but other possibilities include scripting languages or conventional programming languages like Java.

The Serenoa architecture (see figure 1) shows the run-time as a group of three modules:

1. Context Manager
2. Adaptation Engine
3. Run-time Engine

The Context Manager keeps track of the context of use, i.e. information about the user, the device and the environment it is operating in. It provides support for querying the context of use, and for signalling changes.

The Adaptation Engine execute the AAL-DL rules as described above. The Run-time Engine maps the concrete user interface design to the final user interface, in accordance with the adaptations suggested by the Adaptation Engine. The architecture

can be implemented either in the cloud, or in the device itself where the resource constraints permit this.

One challenge is preserving the state of the interaction when applying an adaptation to a change in the context of use. State information can be held at the domain level, the abstract user interface, and the concrete user interface.

Some classes of adaptations can be compiled into the final user interface. For HTML pages, adaptation can be implemented as part of the web page scripts, or through style sheets with CSS Media Queries. This raises the challenge of how to compile high level adaptation rules expressed in AAL-DL into the final user interface.

The Advanced Adaptation Logic Description Language (AAL-DL) seems well suited for standardization, although this may not be practical until we have more experience of how well the run-time architecture performs in a variety of settings.

2.9 Advanced Adaptation Logic Description Language (AAL-DL)

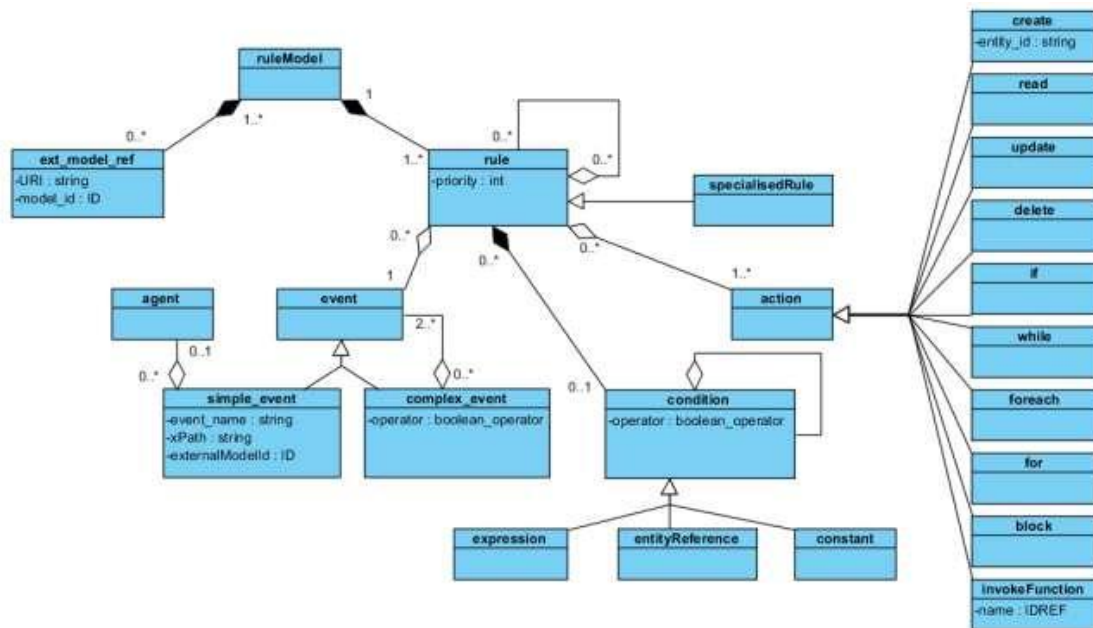
One of the aims of Serenoa is to develop a high-level language for declarative descriptions of advanced adaptation logic [AAL-DL]. This is described in detail in:

- Deliverable D3.3.1 AAL-DL: Semantics, Syntaxes and Stylistics

AAL-DL as currently defined can be used for first order adaptation rules for a specific context of use, and second order rules that select which first order rules to apply. Further work is under consideration for third order rules that act on second order rules, e.g. to influence usability, performance and reliability.

Current examples of AAL-DL focus on adaptation to events signalling changes in the context of use. In principle, it could also be used for design time transformation.

The AAL_DL metamodel is as follows:



This diagram just presents the main subclasses of the action element (create, read, update, delete, if, while, foreach, for, block, and invokeFunction). An XML Scheme has been specified for interchange of AAL-DL rules, but as yet there is not agreement on a high level syntax aimed at direct editing.

Here is an example of a rule:

- *If user is colour-blind then use alternative color palette.*

In XML this looks like:

```

<rule>
  <event>
    <simple_event event_name="onRender"
      XPath="/AbstractUIModel/AbstractInteractionUnit[@id=/AbstractUIModel/@current_unit]"
      externalModelId="uiModel"/>
    </event>
    <condition operator="eq">
      <entityReference XPath="/context/users/user/disability/@blindness"
        externalModelId="ctxModel"/>
      <constant value="colour" type="string"/>
    </condition>
    <action>
      <update>
        <entityReference XPath="/interface/@currentPalette"
          externalModelId="uiModel"/>
        <value>
          <entityReference XPath="/AbstractUIModel/AbstractInteractionUnit/AbstractInteractor[@id='alternativePalette']/@id"
            externalModelId="uiModel" />
        </value>
      </update>
    </action>
  </rule>
</rule>
<rule>
  <event>
    <simple_event event_name="onRender"
      XPath="/AbstractUIModel/AbstractInteractionUnit[@id=/AbstractUIModel/@current_unit]"
      externalModelId="uiModel"/>
    </event>
    <condition operator="eq">
      <entityReference XPath="/context/users/user/disability/@blindness"
        externalModelId="ctxModel"/>
      <constant value="none" type="string"/>
    </condition>
    <action>
      <update>
        <entityReference XPath="/interface/@currentPalette"
          externalModelId="uiModel"/>
        <value>
          <entityReference XPath="/AbstractUIModel/AbstractInteractionUnit/AbstractInteractor[@id='defaultPalette']/@id"
            externalModelId="uiModel"/>
        </value>
      </update>
    </action>
  </rule>
</rule>

```

A significant challenge will be to explore the practicality of enabling developers to work with a high level rule syntax rather than at the level expressed in the XML example.

AAL-DL could be submitted to W3C as a basis for a rule language, however, further work will be needed to demonstrate its practical effectiveness on a range of examples before the W3C Model-Based User Interfaces Working Group is likely to proceed with standardizing an adaptation rule language. In practice, this is something that would likely take place when the Working Group is rechartered in early 2014, i.e. after the Serenoa project comes to an end.

2.10 Are Event-Condition-Action Rules Needed?

The Advanced Adaptation Logic Description Language (AAL-DL) is an example of *event-condition-action* rule languages. The example given above is essentially:

- *when the on-render event occurs and the user is colour blind then set the alternate colour palette*

The use of an event here is redundant if the production rule engine can be configured to run whenever there are changes to the facts used by the rule engine rather than needing to be triggered by events. This is perhaps a matter of the style in which adaptation knowledge is expressed. Some kinds of adaptation can be seen as setting

design choices such as the kind of user interaction control or the colour scheme, e.g. when adapting to changes in ambient lighting. Other kinds of adaptation describe the dialogue between the system and the user, and an event could be critical to triggering a next step in the dialogue such as when changes in the context of use necessitates asking the user for new information, An example could be when the battery level in a mobile device drops below a certain threshold, so that it becomes appropriate to ask the user if she wants to save her work before the power drops too low.

2.11 Abduction and Constraint Satisfaction for Model-Based Design

Conventional solutions for Model-Based UI design provide support for automatic generation of models top-down from higher to lower levels of abstractions. If you alter models at any level other than the highest, you have to manually propagate changes to ensure that the various levels of abstraction are synchronised. This explains the limited appeal of Model-Based UI design as in practice many people find it hard to deal with abstractions, and dislike being forced to design user interfaces top-down. Giving designers the freedom to work how they want remains a critical research challenge for Model-Based UI design!

What kinds of design tools are people looking for? For the Web, we see an increasing range of devices including desktop, smart phones, tablets, connected TVs and cars. In principle, HTML5 can be used on all of these, but the challenge is to adapt to the device and the context of use. The design tools should support distributed teams, and recognize that different team members play different roles with different concerns. An information systems expert is most likely to be interested in the domain and task models. The clients that commissioned the project are likely to want to see mock ups of the user interface, and where practical working examples on real devices. Some people will be happy to work top-down, whilst others prefer to work at the concrete UI level. How can a design tool support these different ways of working?

One approach is that of a collaborative design assistant that attempts to propagate changes made by its users and to reconcile differences when they arise. The design assistant can be implemented in terms of different classes of design rules:

- Rules that propose designs, and which embody design preferences for particular platforms.
- Rules that determine which relationships hold in a given context of use. These rules typically apply across different levels of abstraction.
- Rules that propose changes in response to events signalling changes in the context.
- Rules that critique designs, for example, looking for colour contrasts that would create problems for people who have one of several forms of colour blindness.

Rules that propose designs can start from the domain and task models, and make proposals for the abstract UI, and in turn for the concrete UI. There are relatively few design choices when generating the abstract UI in this way, but many when it comes to generating the concrete UI due to the variety of layouts and available UI controls. A design assistant could rank the designs and present only the best ranked to users for review. If users have previously expressed a preference, this could be used to constrain the search space to just the solutions that are consistent with those preferences. Some changes introduced by users to the concrete UI don't effect the abstract UI, e.g. whether a selection is realized as a drop down menu or a list of radio boxes. Other changes will effect the abstract UI, and some will even alter the domain model, e.g. when changing the set of choices in a drop-down menu.

One approach to propagating the effects of changes across a design is to use event-condition-action rules. When the event occurs the action is triggered if the associated condition hold true. This approach requires every change to be matched with a rule, resulting in the need for large numbers of rules, that then become hard to maintain. A contrasting approach is to express logical relationships across different levels of abstraction. If you know certain facts and also that certain relationships hold true, then it is possible to infer additional facts that must be true if the relationship is to hold. This is generally referred to as abductive reasoning. For simple conjunctive relationships, this can be cast as an extension of relational table joins, using logical variables for values shared across tables. A proof of concept was prepared as an interactive web page. This makes use of a two pass algorithm for performing the logical joins and abducting facts. The demo allows you to enable and disable abduction to see the effects on the results.

- Demo page <http://www.w3.org/2013/01/abduction/>

The constraints fall into two classes: those that are implicit in synchronizing the different levels of abstraction, and those that are imposed by the preferences provided the human designers as they modify the models with the authoring tool. Abduction can also be considered as constraint satisfaction. The design space can be exhaustively scanned to find solutions that fulfil the constraints. This can be time consuming depending on the size of the search space. If no such solution is found, an explanation can be generated to guide the human designer as to why.

The demo cited above uses an exhaustive search which in the worst case is the cross product of the number of records (facts) for all of the tables in the relationship. If the relationship shares variables across the tables in the relationship, the search space diminishes in size. Each table prunes the size of the search space for the next table to consider. This is an example of constraint propagation, which can be used to reduce the size of the search space, but in most cases, you will still have some search left to carry out. A refinement is to use dependency directed backtracking for this search, as this provides a natural basis to generate explanations, see [Petrie1986]. A survey of algorithms for constraint satisfaction problems can be found in [Kumar1992].

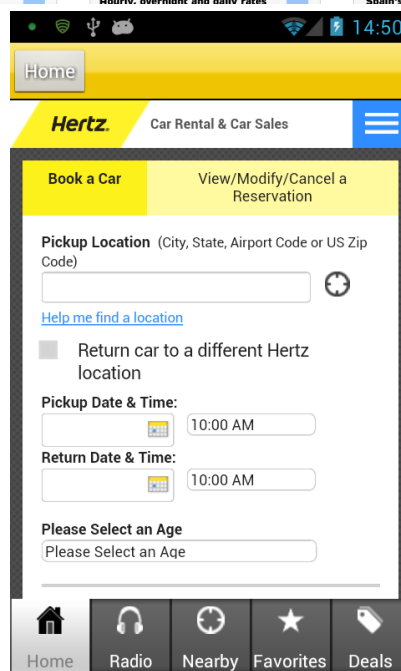
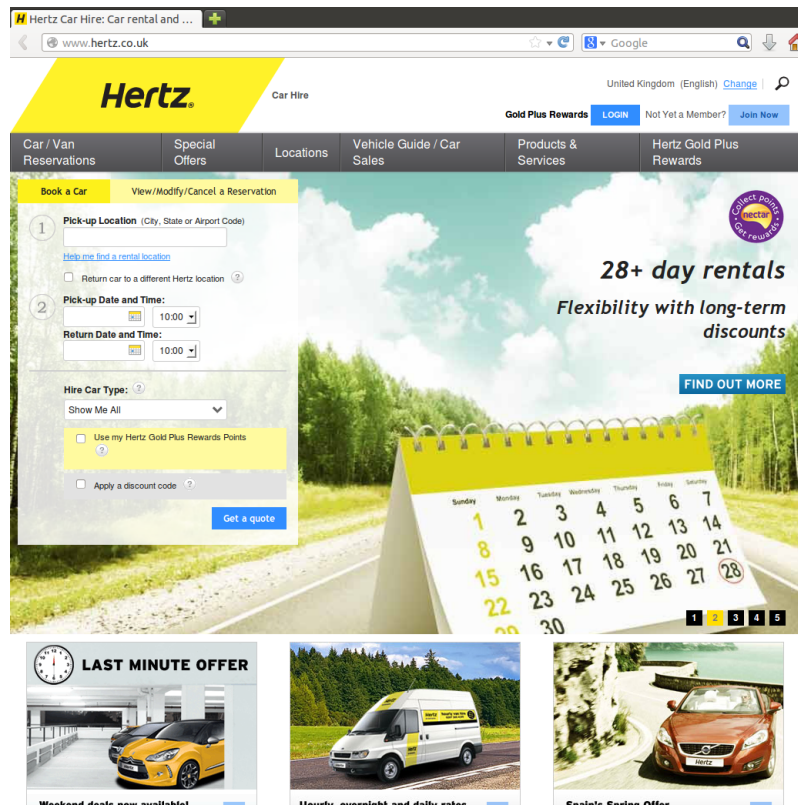
A practical system will involve a mix of techniques. Abduction can be used to update the domain model when the human designer modifies selections. A top-down search

can be used to deal with choices on how to realize abstract UI components at the concrete UI level. Critics can be used to identify poor choices.

For more details, see [Raggett2013] and [Motti2013].

2.11.1 How do user interfaces vary across different devices?

On a mobile device with a small screen users are likely to be more task focused compared with sitting at a desk with a much larger display and a full keyboard. For a design assistant to take this into account, there needs to be appropriate annotations at the domain and task level. Here are some motivating examples.



The two screen shots above were taken from the Hertz car rental company website and their Android application respectively. The comparison illustrates the difference in the user experience on desktop and smart phones. The larger desktop screen allows for a richer experience with additional marketing opportunities compared to the smaller smart phone display. Note also that Hertz chose to add smart phone specific capabilities such as “Radio” and “Nearby” via a toolbar at the bottom of the screen. The task models could be annotated to indicate a task's suitability for particular contexts of use.

2.12 Corporate Rules for Consistent User Experience

Companies often wish to ensure that the user interfaces on their products have a consistent look and feel that expresses the brand the company is promoting. It is still the case that many designers focus first on the visual appearance by working with tools like Adobe Illustrator to mock up the appearance of a user interface. This leads to costly manual processes for reviewing whether the resultant user interface designs match corporate standards.

The Serenoa Framework has the potential to make this a lot easier through the separation of design concerns and the application of design and run-time rule engines. The rules can be written to verify adherence to corporate standards as the user interface is being designed. At run-time, business rules can be used to implement corporate standards. The concrete user interface languages can be designed to reflect the templates and components required. The process of transforming the concrete user interface into the final user interface can be designed to apply the corporate branded look and feel (skinning the user interface).

Further work is needed to identify what changes are needed to support this in the rule language, and its suitability for standardization. There is some potential for standardizing the means for skinning the concrete user interface for particular classes of target platforms.

3 Prospective Standards Development Organizations

Serenoa focuses on model-based techniques for context aware user interface design. The final user interface is generated for the target platform, e.g.

- The Open Web Platform (HTML5)
- Native apps on smart phones and tablets (e.g. iOS, Android, Windows 8)
- Hybrid apps on smart phones and tablets that combine native and Web technologies, e.g. [PhoneGap]
- Feature phones with [J2ME] and [BREW]
- Java apps on desktop computers

Further possibilities include connected TVs and cars, and specialized user interfaces in aeroplanes, and on the factory floor. When it comes to standardization the most obvious organization is the World Wide Web Consortium (W3C) which seeks to develop standards for Web technologies across a broad range of devices. The scope covers markup language such as HTML, SVG and XML, style sheet languages (CSS), and scripting APIs for rich access to device capabilities. One of the advantages of W3C is its commitment to ensuring royalty free implementations of its standards. Other organizations typically only require licensing of intellectual property rights (IPR) under fair, reasonable and non-discriminatory terms (FRAND).

European research projects generally favour the Java programming language along with the Eclipse integrated development environment (IDE). This suggests exploring the potential for exploiting the Java Community Process (JCP). This is a forum for

developing standard technical specifications for Java technology. In principle, Serenoa partners could submit a proposal for a new Java Specification Request (JSR). However, the JCP is normally used for developing Java language APIs rather than the meta-models and adaptation rule language being researched by the Serenoa project. As a result, this avenue for standardization has not been followed up. Other possible standards development organizations include [ECMA] which is well known for the JavaScript language standard (ECMAScript), [ETSI] which focuses on telecommunication standards, [OASIS] which focuses on e-business and web services, the OMG which focuses on enterprise integration standards, and the IETF which focuses on core protocols for the Internet. None of these seemed like a good match to the needs of the Serenoa project. The same applies for the ISO given the overheads of setting up and participating in an ISO technical committee.

3.1 W3C Organization and Process

[W3C] is an international member-based organization hosted by MIT in North America, ERCIM in France, Keio University in Japan and Beihang University in China. Funding is from W3C Member fees, research grants and other sources of private and public funding, sponsorship and donations. W3C's standards track work takes place in its Working Groups where representatives from W3C Member organizations together with invited experts work on drafting and refining specifications according to the [W3C Process]. This has the following steps:

1. Member submissions
2. Editor's draft
3. First Public Working Draft
4. Updated Working Draft
5. Last Call Working Draft
6. Candidate Recommendation
7. Proposed Recommendation
8. W3C Recommendation

Public Working Drafts are snapshots of work in progress that the Working Group deems appropriate for public review. A Last Call Working Draft is published when the Working Group believes that the specification is stable and is usually the time when the most feedback is received, either from other W3C Working Groups or from people external to the W3C. A Candidate Recommendation is issued along with a call for implementations, and is the point where W3C seeks confirmation that the specification is implementable, and has sufficient industry support to ask the W3C Advisory Committee to review it as a Proposed Recommendation. The final step is a W3C Recommendation, W3C's term for its standards.

At various points along the way, W3C Member organizations can exercise their right to opt out particular technologies when they are unwilling to license it on a royalty free basis. This triggers a W3C Patent Advisory Group to provide advice on how to proceed. In some cases, this has resulted in a call for prior art. Patent Advisory Groups may also be triggered by non W3C Members asserting their patents. As far as

is known, the specifications developed by the Serenoa project are unencumbered by patents, which is good news for the prospects for standardization work.

In addition to Working Groups, W3C also has Community Groups and Business Groups. These are intended for pre-standardization activities, such as community building, collecting use cases and requirements, and preparation of draft specifications with a view to transferring them into a W3C Working Group.

Serenoa project partners which are W3C Members (other than W3C itself):

- Telefónica I+D
- Université catholique de Louvain (UCL)
- Consiglio Nazionale delle Ricerche / Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo"
- SAP Research
- Fundación CTIC-Centro Tecnológico (CTIC-CT)

All of the above are involved in the W3C Model-Based UI Working Group with the exception of SAP.

4 W3C Model-Based UI Working Group

This section of the report describes standardization activities at the W3C on model-based user interface design.

4.1 MBUI WG - Introduction

The W3C Model Based User Interfaces Working Group was formed on 17 October 2011 and provides the main target for standardizing work from the Serenoa project. This section will describe the history leading up to the formation of this Working Group, its charter, the technical submissions received, the current work items and future plans.

The Working Group conducts its business with a mix of email, regular phone conferences and face to face meetings. The most active members of the group include:

- Paolo Bottoni, University of Rome
- Heiko Braun, Red Hat
- Cristina Gonzalez Cachon, Fundación CTIC, Asturias
- Gaëlle Calvary, Laboratory of Informatics of Grenoble
- Joëlle Coutaz, Laboratory of Informatics of Grenoble
- Sebastian Feuerstack, Universidade Federal de São Carlos, São Paulo
- Nikolaos Kaklanis, Centre for Research and Technology Hellas
- Carlos Laufer, Pontifical Catholic University of Rio de Janeiro
- Gerrit Meixner, Heilbronn University
- Vivian Genaro Motti, Université catholique de Louvain
- Fabio Paternò, ISTI, Consiglio Nazionale delle Ricerche, Pisa
- Jaroslav Pullmann, Fraunhofer Gesellschaft

- Dave Raggett, W3C/ERCIM
- Javier Rodríguez Escolar, Fundación CTIC, Asturias
- Carmen Santoro, ISTI, Consiglio Nazionale delle Ricerche, Pisa
- Davide Spano, ISTI, Consiglio Nazionale delle Ricerche, Pisa
- Jean Vanderdonckt, Université catholique de Louvain

A full list is available at

- <http://www.w3.org/2000/09/dbwg/details?group=50171&public=1>

4.2 MBUI WG History

When Tim Berners-Lee invented the World Wide Web at the start of the nineties, he set out to ensure that it would be accessible from a wide range of platforms. Early examples include the NeXT computer, a sophisticated graphics workstation, and dumb text terminals using the CERN Line Mode Browser. By the mid-nineties, popular browsers included Netscape's Navigator, and Microsoft's Internet Explorer. The success of the latter meant that most people were interacting with the Web from a desktop computer running Microsoft Windows. Some websites even went as far as stating "best viewed in Internet Explorer".

By the end of the nineties, as the potential of mobile devices began to get people's attention, the challenge arose for how to enable designers to create Web applications for use on desktop and mobile devices. W3C launched the Device Independence Working Group to address these challenges. A set of draft device independence principles were published in September 2001:

- <http://www.w3.org/TR/2001/WD-di-princ-20010918/>

This followed on from earlier work at W3C on Composite Capability/Preference Profiles (CC/PP), a means for devices to advertise their capabilities so that web sites could deliver content adapted to the needs of each device. That led to a W3C Recommendation for CC/PP 1.0 in January 2004:

- <http://www.w3.org/TR/2004/REC-CCPP-struct-vocab-20040115/>

W3C went on to work on a device independent authoring language (DIAL). This combines HTML with simple rules according to the device's capabilities.

- <http://www.w3.org/TR/2007/WD-dial-20070727/>

With DIAL, adaptation could take place anywhere along the delivery chain, i.e. at the originating web site, a proxy server or in the browser. CC/PP and DIAL both failed to take off in practice. One issue was that mobile device vendors failed to provide accurate information on device capabilities. Another was browser developers had at that time little interest in device independence, with the exception of limited support for conditionals in style sheets (CSS Media Queries):

- <http://www.w3.org/TR/CSS2/media.html>

This allowed you to provide different style rules for a limited set of device categories:

- **all** - Suitable for all devices.
- **braille** - Intended for braille tactile feedback devices.
- **embossed** - Intended for paged braille printers.
- **handheld** - Intended for handheld devices (typically small screen, limited bandwidth).
- **print** - Intended for paged material and for documents viewed on screen in print preview mode. Please consult the section on paged media for information about formatting issues that are specific to paged media.
- **projection** - Intended for projected presentations, for example projectors. Please consult the section on paged media for information about formatting issues that are specific to paged media.
- **screen** - Intended primarily for color computer screens.
- **speech** - Intended for speech synthesizers. Note: CSS2 had a similar media type called 'aural' for this purpose. See the appendix on aural style sheets for details.
- **tty** - Intended for media using a fixed-pitch character grid (such as teletypes, terminals, or portable devices with limited display capabilities). Authors should not use pixel units with the "tty" media type.
- **tv** - Intended for television-type devices (low resolution, color, limited-scrollability screens, sound available).

Few browsers supported CSS media queries apart from screen and print. More recently, the specification has added further capabilities and finally became a W3C Recommendation in June 2012.

- <http://www.w3.org/TR/2012/REC-css3-mediaqueries-20120619/>

A further possibility is to use web page scripts to adapt the markup and presentation locally in the browser. Each browser provides the user agent string, but by itself this doesn't provide sufficient information for effective adaptation. The scripting APIs for accessing information about the device are extremely limited. In part this is driven by concerns over privacy. The more information a website can determine about a device, the easier it is to fingerprint a user and to build up a detailed picture of the user's browsing habits.

DIAL, CSS Media Queries, and client side scripting all fail to tackle the challenge of separating out different level of design concerns. This is where research work on model-based user interface design has the most promise. The next sections will describe how this was picked up by W3C, and the launch of the Model-Based User Interfaces Working Group.

4.2.1 MBUI Incubator Group

W3C work on model-based user interfaces started with a preliminary meeting in Pisa, Italy on 23 July 2008, hosted by the Istituto di Scienze e Tecnologie

dell'Informazione, and concluded with the participants agreeing to work together on preparing a draft charter for a W3C Incubator Group.

- <http://www.w3.org/2008/07/model-based-ui.html>

The first face to face meeting of the Model-Based User Interfaces Incubator Group was held on 24 October 2008, hosted by W3C at the 2008 Technical Plenary in Mandelieu, France. The Charter and home page for the Model-Based Interfaces Incubator Group can be found at:

- <http://www.w3.org/2005/Incubator/model-based-ui/charter/>
- <http://www.w3.org/2005/Incubator/model-based-ui/>

Work proceeded via teleconferences and a wiki. A second face to face meeting took place in Brussels on 11-12 June 2009, hosted by the Université catholique de Louvain. The Incubator Group report was published on 4 May 2010.

- <http://www.w3.org/2005/Incubator/model-based-ui/XGR-mbui-20100504/>

It provides an introduction to model-based UI design, a survey of the state of the art, an outline of motivating use cases, and a case study of user interfaces in the digital home. The concluding remarks cover suggested standardization work items.

The publication of the Incubator Group report was followed by a Workshop in Rome. This is described in the next section.

4.2.2 MBUI Workshop

The W3C Workshop on Future Standards for Model-Based User Interfaces was held on 13-14 May 2010 in Rome, hosted by the Istituto di Scienze e Tecnologie dell'Informazione. The website includes the statements of interest submitted by participants, the agenda and links to talks, and the Workshop Report, which can be found at:

- <http://www.w3.org/2010/02/mbui/report.html>

The Workshop was timed to follow the publication of the report of the W3C Model-Based UI Incubator Group. Participants presented model-based approaches from a variety of perspectives, reflecting many years of research work in this area. The Workshop's final session looked at the opportunity for launching standards work on meta-models as a basis for exchanging models between different markup languages.



Figure 4: Participants of the Rome 2010 MBUI Workshop

4.2.3 Formation of MBUI Working Group

The W3C Model-Based User Interfaces Working Group was launched on 17 October 2011 and is chartered until the 13 November 2013. Work is proceeding with a mix of regular teleconferences, the mailing list and wiki, and face to face meetings. The first face to face was hosted by DFKI in Kaiserslautern, Germany on 9-10 February 2012,



and the second on 14-15 June in Pisa, hosted by ISTI-CNR.



and the fourth on 29-30 October 2012 as part of the W3C Technical Plenary in Lyon.



and the fifth on 10-11 July in Grasbrunn, near Munich, hosted by Red Hat



4.3 MBUI Working Group Charter

- <http://www.w3.org/2011/01/mbui-wg-charter>

The Working Group Charter defines the scope of the permitted work items, and the roadmap as envisioned when the charter came into effect. The charter is subject to review by the W3C Advisory Committee, which includes one person per member organization (regardless of size), and the W3C Management Team. The Model-Based User Interfaces (MBUI) Working Group is currently chartered until 30 November 2013. The scope is as follows:

Use cases and Requirements

As needed to guide and justify the design decisions for the development of the specifications.

Specification of meta-models for interchange of models between authoring tools for (context aware) user interfaces for web-based interactive application front ends

This could take the form of UML diagrams and OWL ontologies, and cover the various levels of abstraction (e.g. as defined in the Cameleon reference framework, as well as that needed to support dynamic adaption to changes in the context).

Specification of a markup language and API which realize the meta-models.

This is expected to draw upon existing work such as (but not restricted to) Concur Task Trees (CTT), Useware Markup Language (useML), UsiXML or UIML.

Test assertions and Test suite for demonstrating interoperability

This is needed to support progress along the W3C Recommendation Track, and in particular, to exit from the Candidate Recommendation phase.

Model-based user interface design primer

An explanation/guideline for how to apply the specifications to support the development of the associated use cases.

Open Source Implementations

Working Group members may wish to develop open source implementations of authoring tools to demonstrate the potential, and for use in developing and applying the test suite described above.

Some features are explicitly **out of scope** for the Working Group

Defining markup and APIs for direct interpretation by interactive application front ends (e.g. web browsers).

But where appropriate, it should be feasible to define markup, events and APIs that are supported by libraries, e.g. JavaScript modules. This may be needed to support dynamic adaptation to changes in the context.

This restriction was included in the charter to reassure browser vendors that there is no requirement for changes to Web browsers. Instead, the work on model-based user interface design is aimed at authoring tools, and associated run-time libraries that run on top of browsers.

4.3.1 Work Items

The Charter identifies the following deliverables:

- Recommendation Track specification for task models
- Recommendation Track specification for abstract user interface models
- Working Group Note introducing model based user interface design, along with use cases
- Working Group Note defining a glossary of terms as used in the other deliverables

W3C Recommendation Track specifications follow the following stages. This have been annotated with the dates the MBUI deliverables were envisioned by the charter to reach each stage.

1. **First Public Working Draft** - initial publication (expected March 2012)

2. **Last Call Working Draft** - stable version (expected September 2012)
3. **Candidate Recommendation** - test suites and implementation reports (expected February 2013)
4. **Proposed Recommendation** - reviewed by W3C Advisory Committee (expected June 2013)
5. **Recommendation** - supplemented by errata (expected August 2013)

In the preparatory work leading up to drafting the charter, there was general agreement that it would be best to focus initial work on standards for task models and abstract UI models. Once this has been achieved, the next step will be to work on standards for concrete UI models, and context adaptation. This would require re-chartering the MBUI Working Group for a further period. Further details will be discussed in the conclusion to this report.

4.4 MBUI Submissions

When the Model-Based User Interfaces Working Group was formed, the first step was to invite submissions of background work as a basis for discussions leading to a consensus on the specifications we planned to create. There were 7 submissions by the time we met for the first face to face meeting in Kaiserslautern. The following subsections briefly review each in turn. Further information can be found on the MBUI Wiki at:

- http://www.w3.org/wiki/MBUI_Submissions

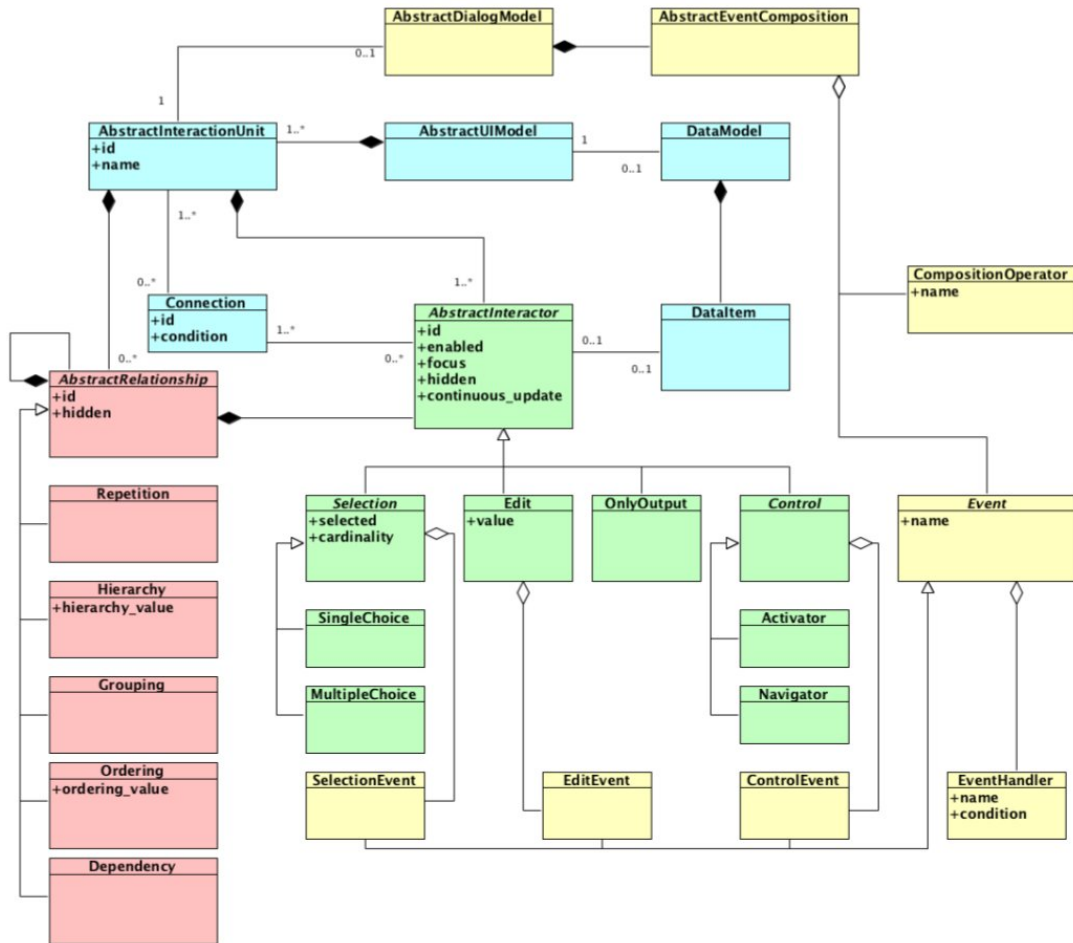
4.4.1 Advanced Service Front-End Description Language (ASFE-DL)

- <http://www.w3.org/2012/01/asfe-dl/>

This is a submission on behalf of the FP7 Serenoa project, and covers a meta-model and XML serialization for the abstract UI layer of the Cameleon Reference Framework, see:

- CAMELEON (Context Aware Modelling for Enabling and Leveraging Effective interaction) Project (FP5-IST4-2000-30104), <http://giove.isti.cnr.it/projects/cameleon.html>.

The ASFE-DL language is expected to evolve further during the remainder of the Serenoa project, but the version submitted just focuses on abstract user interface models, and corresponds to the Platform-Independent-Model – PIM in Model Driven Engineering (MDE). ASFE-DL draws upon experience with previous work on MARIA and UsiXML, both of which were submitted separately to the MBUI Working Group. The idea behind ASFE-DL is to create a unified and more complete language, combining the strengths of the two languages, unifying concepts and adding new features that will allow this language to meet requirements for context aware adaptation of service front ends. The ASFE-DL meta-model (for the submission) is defined by the following UML diagram:



Different colours are used to highlight different parts of the metamodel: sky-blue for the main structure of the interface, green for the interactor hierarchy, red for the classes that model the relationships between interactors and yellow for the classes that model the UI behaviour.

Loosely put, ASFE-DL can be used to describe the user interface as a set of interrelated abstract dialogues (AbstractInteractionUnits), where each dialogue has a set of interactors for collecting user input, updating the domain model, activating methods on the domain model, and navigating between dialogues. ASFE-DL provides a means to define handlers for a variety of events, which can be triggered by user actions, or by the system itself.

4.4.2 The ConcurTaskTrees Notation (CTT)

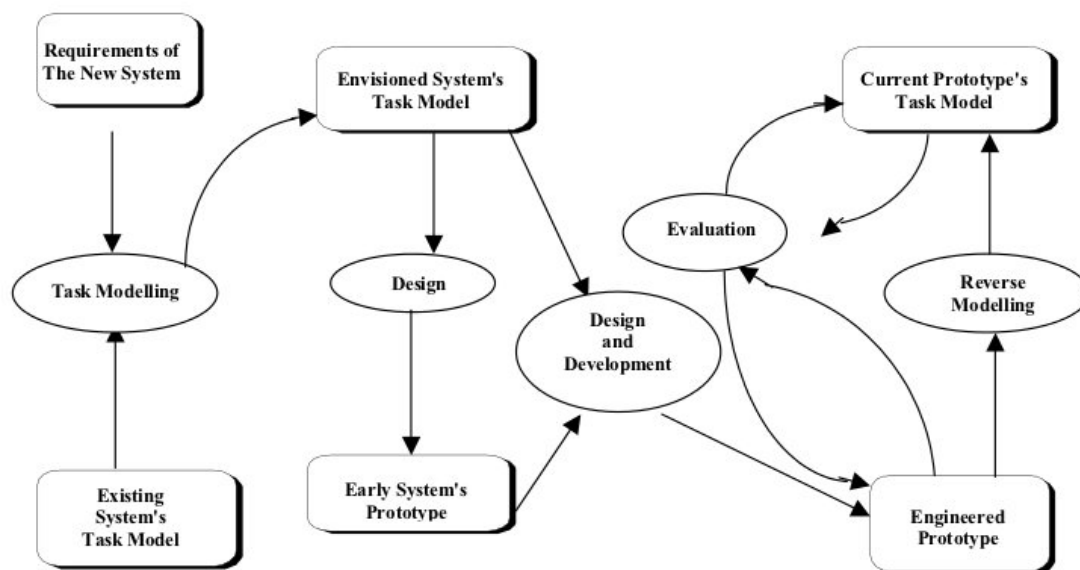
* <http://www.w3.org/2012/02/ctt/>

The ConcurTaskTrees (CTT) notation provides a metamodel, visualization and XML format for interchange of user interface task models between different design tools. CTT was developed by ISTI-CNR and first published at INTERACT'97 and since then has been widely used in academic and industrial institutions.

Task models can be used in a variety of ways:

- Improve understanding of the application domain
- Record the result of interdisciplinary discussion
- Support effective design
- Support usability evaluation
- Support the user during a session
- Documentation

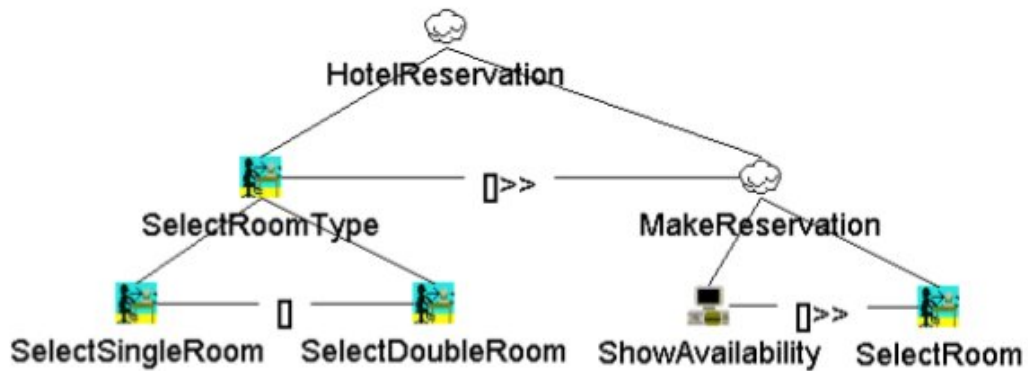
The aim of CTT is to provide fairly high level descriptions of user interfaces. It is not intended as a programming language, and deliberately omits details that would risk derailing high level design discussions. Extensions have been proposed for cooperative task models involving multiple users.



The notation covers:

- Hierarchical structuring of tasks
- Temporal relations between tasks
- Task allocation (user or system)
- Task preconditions

CTT task models are frequently depicted as a diagrams, e.g.



The temporal operators are as follows:

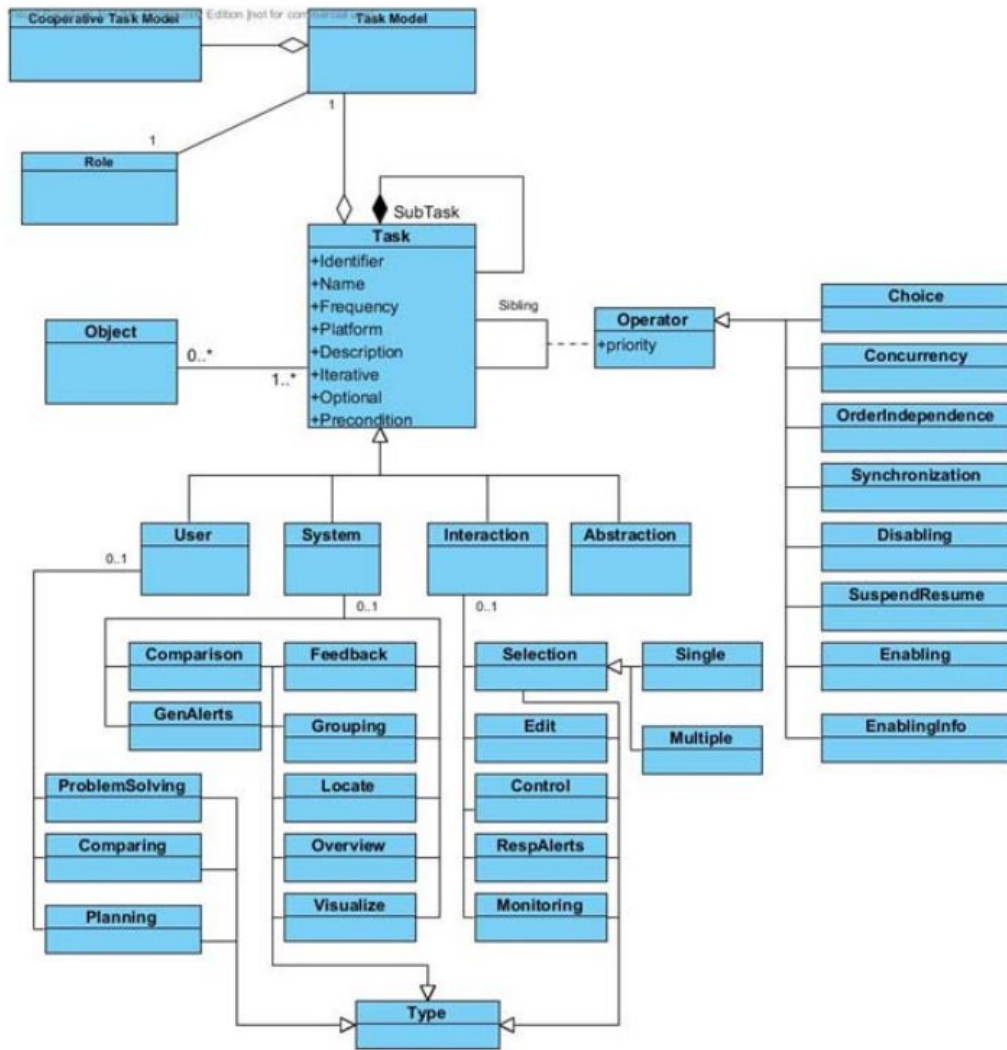
Operator	Symbol
Enabling	T1 >> T2 or T1 []>> T2
Disabling	T1 [> T2
Interruption	T1 > T2
Choice	T1 [] T2
Iteration	T1* or T1 {n}
Concurrency	T1 T2 or T1 [] T2
Optionality	[T]
Order Independency	T1 = T2

Where the second symbol for enabling is for task enabling with information passing. Likewise, the second symbol for concurrency is for concurrent communicating tasks.

Tasks can be allocated as follows

- **System** - data presentation or action carried out by the system
- **User input** - data entry by the user
- **Cognition** - a cognitive task carried out by the user

CTT's meta-model as a UML diagram:



There is also an XML schema to support interchange of models in the XML format.

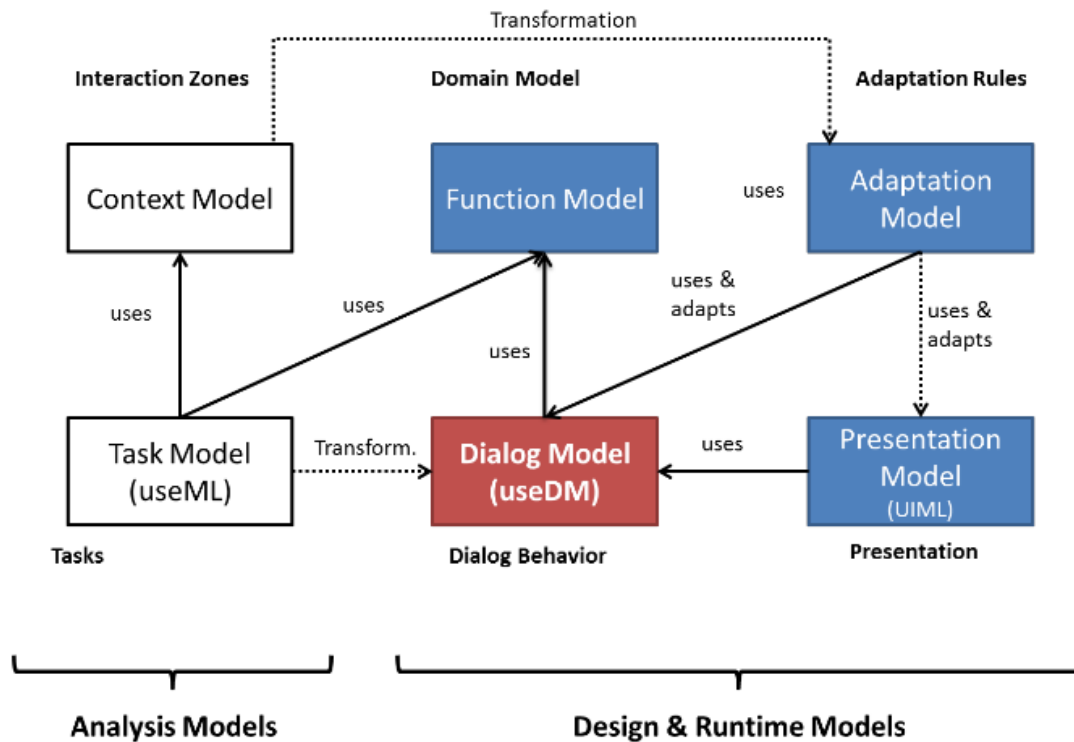
4.4.3 Useware Markup Language (UseML)

- [http://www.w3.org/wiki/Useware_Markup_Language_\(UseML\)](http://www.w3.org/wiki/Useware_Markup_Language_(UseML))

The Useware Markup Language (UseML) and dialog modelling language (UseDM) have been developed to support the user and task oriented Useware Engineering process and has been applied to the domain of production automation and industrial environments. The Useware process has the following steps:

1. Analysis
2. Structuring
3. Design
4. Realization

The following diagram illustrates the various kinds of models involved:

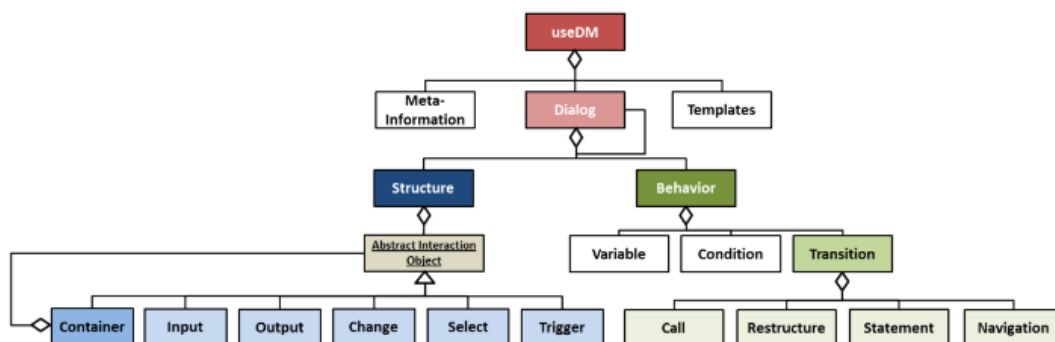


The use model abstracts platform-independent tasks, actions, activities, and operations into use objects that make up a hierarchically ordered structure. Each element of this structure can be annotated by attributes such as eligible user groups, access rights, importance. Use objects can be further structured into other use objects or elementary use objects. Elementary use objects represent the most basic, atomic activities of a user, such as entering a value or selecting an option

Currently, five types of elementary use objects exist:

- Trigger: starting, calling, or executing a certain function of the underlying technical device (e.g., a computer or field device)
- Select: choosing one or more items from a range of given ones
- Input: entering an absolute value, overwriting previous values
- Output: the user gathers information from the user interface
- Change: making relative changes to an existing value or item

The following diagram describes the UseDM meta-model:



The presentation model covers the layout and style aspects for the elements given in the dialogue model. The presentation model is specified using the User Interface Markup Language (UIML), which is covered in the following subsection of this report.

4.4.4 User Interface Markup Language (UIML)

This was an indirect submission, as UIML is the presentation language for UseML. [UIML] was developed by Marc Abrams, et al. in the late 1990's to address the challenges of developing for a growing variety of target devices for user interfaces.

UIML is an XML language for implementing user interfaces, see [UIML]. It combines appliance independent presentation concepts with appliance dependant concepts. Please refer to the following link for a discussion of the relationship of UIML to other interface description languages:

- <http://www.oasis-open.org/committees/download.php/3419/The%20Relationship%20of%20the%20UIML%203%20v01.03.doc>

Here is a pertinent extract:

UIML was not intended as a UI design language, but rather as a language for UI implementation. Therefore UI design tools could represent a design in a design language, and then transform a UI in a design language to a canonical representation for UI implementation, namely UIML.

UIML has been standardized by OASIS, see:

- https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=uiml

UIML describes a user interface with five sections: description, structure, data, style, and events. The template looks like:

```
<?xml version="1.0" standalone="no"?>
<uiml version="2.0">

  <interface name="..." class="MyApps">
    <description>...</description>
    <structure>...</structure>
    <data>...</data>
    <style>...</style>
    <events>...</events>
  </interface>

  <logic>
  </logic>
```


</uiml>

The description element assigns a name and a class to each UI component. The structure element defines which components are present from the description, and how they are organized as a hierarchy. The data element binds to application dependent data. The style element binds UI components to their implementation, e.g. java classes such as "java.awt.MenuItem". The events element binds events to actions. You can use application dependent, but appliance independent events, and then bind them to appliance dependent events through the style element. OASIS is currently working on version 4 of the UIML specification.

An longer introduction to UIML can be found at:

- <http://www8.org/w8-papers/5b-hypertext-media/uiml/uiml.html>

4.4.5 Abstract Interactor Model (AIM) Specification

- <http://www.multi-access.de/mint/aim/2012/20120516/>

AIM focuses on modelling multimodal interactions in terms of modes and media.

The three basic interactors are:

1. Abstract Interactor Model describing behaviour common to all modes and media
2. Concrete Interactor Model describing user interface for a certain mode or medium
3. Interaction Resource Model - a database used to store and manage interactor state

The following figure shows the interactor class and its relations to the three basic interactors as a UML diagram:

AIM has been implemented using a range of web technologies: WebSockets, HTML5 / CSS3, Rails, NodeJS, Redis/TupleSpace, and MMI-Arch. For more details see the link above.

4.4.6 Multimodal Interactor Mapping (MIM) Model Specification

- <http://www.multi-access.de/mint/mim/2012/20120203/>

This submission supplements the submission on Abstract Interactor Model (AIM) Specifications.

Multimodal Mappings

Each multimodal mapping consists of:

- **Observations** - used to observe state charts (state machines) for state changes
- **Actions** - used to trigger state changes by sending events to start charts or to call functions in the backend
- **Operators** - specify multimodal relations and link a set of observations to a set of actions

There are six operators: sequence, redundancy, complementary, assignment, and equivalence.

Synchronization Mappings

These are predefined together with interactors.

Exemplary Mappings

- Drag and drop
- Gesture based navigation

4.4.7 UsiXML

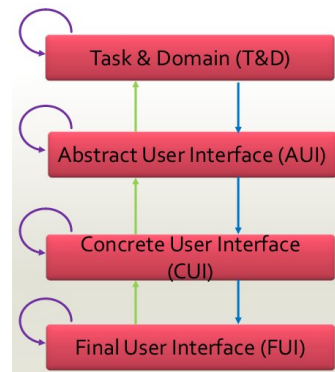
- http://www.w3.org/wiki/images/5/5d/UsiXML_submission_to_W3C.pdf
- <http://www.w3.org/wiki/images/8/83/UsiXMLSubmission-Kaiserslautern-Feb2012-Part1.pdf>
- <http://www.w3.org/wiki/images/3/3e/UsiXMLSubmission-Kaiserslautern-Feb2012-Part2.pdf>
- <http://www.w3.org/wiki/images/9/96/UsiXMLSubmission-Kaiserslautern-Feb2012-Part3.pdf>

The User Interface eXtensible Markup Language (UsiXML) is a XML-compliant markup language that describes the user interface for multiple contexts of use such as Character User Interfaces (CUIs), Graphical User Interfaces (GUIs), Auditory User

Interfaces, and Multimodal User Interfaces. UsiXML has been defined by the UsiXML Consortium, see:

- <http://www.usixml.org/>

The semantics are defined in a UML 2.0 class diagram, MOF-XMI and as an ontology using OWL-Full 2.0. The interchange syntax is XML and defined with an XML schema. UsiXML is based upon the CAMELEON Reference Framework:



Where task models can be defined with the ConcurTaskTree (CTT) notation and mapped to abstract user interface models (independent of devices and modalities), and thence to concrete user interface models (designed for a class of devices and modalities) and compiled into a final user interface for delivery to specific device platform. Domain models describe the interface to the user interface back end in terms of properties and methods that can be invoked based upon user interaction. Behaviour can be described in terms of event driven state transition models using W3C's State Chart XML (SCXML).

You can define different kinds of mappings:

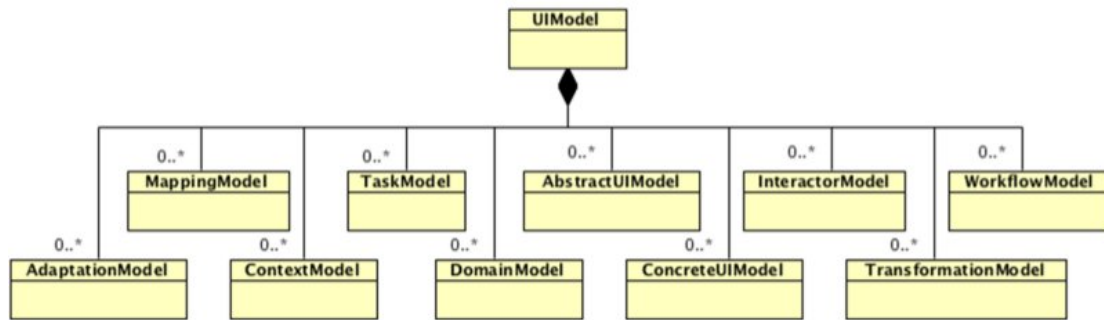
- Reification: from high to lower-level
- Abstraction: from low to higher-level
- Reflexion: at the same level

Reflexion is useful for transcoding, graceful degradation, restructuring and retasking.

UsiXML defines context of use models

- User models, e.g. personal preferences and abilities
- Platform models, e.g. device capabilities
- Environment models, e.g. ambient light and noise

The UsiXML metamodel is as follows:



UsiXML is accompanied with a plugin for the Eclipse Integrated Development Environment.

Proposed UsiXML extension enabling the detailed description of the users with focus on the elderly and disabled

- http://www.w3.org/wiki/images/f/f8/An_extension_of_UsiXML_for_users.pdf

This introduces a unified user modelling technique designed to support user interfaces for the elderly and disabled.

Two new models are proposed for UsiXML's uiModel:

- **disability model**
- **capability model**

This covers the relationship between affected tasks and various kinds of disabilities including both physical and cognitive disabilities.

4.4.8 MARIA

MARIA (Model-based language for Interactive Applications) [Paterno2000], is a universal, declarative, multiple abstraction-level, XML-based language for modelling interactive applications in ubiquitous environments.

MARIA supports the CAMELEON framework, with one language for the abstract description (the so-called “Abstract User Interface” level, in which the UI is defined in a platform –independent manner) and multiple platform-dependent languages (which are at the level of the so-called “Concrete User Interface”), which refine the abstract one depending on the interaction resources at hand . Examples of platforms are the graphical desktop, the graphical mobile, the vocal platform, etc.

Abstract User Interface

The Abstract User Interface (AUI) level describes a UI only through the semantics of the interaction, without referring to a particular device capability, interaction modality or implementation technology

At the abstract level, a user interface is composed of a number of presentations, has an associated data model, and can access a number of external functions. Each presentation is composed of a number of interactors. (basic interaction elements) and a set of interactor compositions.

According to its semantics an interactor belongs to one the following subtypes:

- **Selection.** Allows the user to select one or more values among the elements of a predefined list. It contains the selected value and the information about the list cardinality. According to the number of values that can be selected, the interactor can be a Single Choice or a Multiple Choice.
- **Edit.** Allows the user to manually edit the object represented by the interactor, which can be text (Text Edit), a number (Numerical Edit), a position (Position Edit) or a generic object (Object Edit).
- **Control.** Allows the user to switch between presentations (Navigator) or to activate UI functionalities (Activator).
- **Only output.** Represents information that is submitted to the user, not affected by user actions. It can be a Description that represents different types of media, an Alarm, a Feedback or a generic Object.

The different types of interactor-compositions are:

- **Grouping:** a generic group of interactor elements.
- **Relation:** a group where two or more elements are related to each other.
- **Composite Description:** represents a group aimed to present contents through a mixture of Description and Navigator elements.
- **Repeater** which is used to repeat the content according to data retrieved from a generic data source.

MARIA XML allows describing not only the presentation aspects but also the behaviour Data Model. The interface definition contains description of the data types that are manipulated by the user interface. The interactors can be bound with elements the data model, which means that, at runtime, modifying the state of an interactor will change also the value of the bound data element and vice-versa. The main features available already at the abstract level and common to all languages are:

- **Data Model.** The interface definition contains description of the data types that are manipulated by the user interface. The interactors can be bound with elements the data model, which means that, at runtime, modifying the state of an interactor will change also the value of the bound data element and vice-versa. This mechanism allows the modelling of correlation between UI elements, conditional layout, conditional connections between presentations,

input values format. The data model is defined using the standard XML Schema Definition constructs.

- **Generic Back End.** The interface definition contains a set of External Functions declarations, which represents functionalities exploited by the UI but implemented by a generic application back-end support (e.g. web services, code libraries, databases etc.). One declaration contains the signature of the external function that specifies its name and its input/output parameters.
- **Event Model.** Each interactor definition has a number of associated events that allow the specification of UI reaction triggered by the user interaction. Two different classes of events have been identified: the Property Change Events that specify the value change of a property in the UI or in the data model (with an optional precondition), and the Activation Events that can be raised by activators and are intended to specify the execution of some application functionalities (e.g. invoking an external function).
- **Continuous update of fields.** It is possible to specify that a given field should be periodically updated invoking an external function.
- **Dynamic Set of User Interface Elements.** The language contains constructs for specifying partial presentation updates (dynamically changing the content of entire groupings) and the possibility to specify a conditional navigation between presentations. This set of new features allows having already at the abstract level a model of the user interface that is not tied to layout details, but it is complete enough for reasoning on how UI supports both the user interaction and the application back end.

Concrete User Interface

A Concrete User Interface (CUI) in MARIA XML provides platform-dependent but implementation language-independent details of a UI. A platform is a set of software and hardware interaction resources that characterize a given set of devices. MARIA XML currently supports the following platforms:

- Desktop CUI s model graphical interfaces for desktop computers.
- Mobile CUI s model graphical interfaces for mobile devices.
- Multimodal Desktop CUI s model interfaces that combine the graphical and vocal modalities for desktop computers.
- Multimodal Mobile CUI s model interfaces that combine the graphical and vocal modalities for mobile devices.
- Vocal CUI s interfaces with vocal message rendering and speech recognition.

Each platform meta-model is a refinement of the AUI, which specifies how a given abstract interactor can be represented in the current platform. The followings paragraphs provide a brief description of the Desktop CUI and of the Vocal CUI.

Concrete Desktop User Interface

A CUI meta-model for a given platform is an extension of the AUI meta-model, which means that all the entities in the AUI still exist in the CUI. The extensions add the platform-dependent information (but still implementation language independent) to the structure of the corresponding AUI model for the same application interface by either adding attributes or extending through an inheritance mechanism the existing entities for the specification of the possible concrete implementation of the abstract interactors. In this paragraph, we will introduce the extension to the AUI meta-model for the definition of the Graphical Desktop CUI meta-model. The existing elements with new attributes are: **Presentation**: it contains the `presentation_setting` attribute, which contains information on the title, background (color or image) and the font used. **Grouping**: it contains the `grouping_setting` attribute, which contains the information on the grouping display technique (grid, fieldset, bullet, background color or image) and if the elements are related with an ordering or hierarchy relation. The classes which have been extended using the inheritance are the following:

- An **Activator** can be implemented as a *button*, a *text_link*, *image_link*, *image_map* (an image with the definition of a set of areas, each one associated with a different value) or *mailto*
- An **Alarm** can be implemented as a *text* (a text with font and style information) or an *audio_file*
- A **Description** can be implemented as a *text*, *image*, *audio*, *video*, *table*
- A **MultipleChoice** can be implemented as a *check_box* or a *list_box*
- A **Navigator** can be implemented as an *image_link*, *text_link*, *button*, *image_map*.
- A **NumericalEditFull** can be implemented as a *text_field* or a *spin_box* (a text field which includes also up and down buttons)
- A **NumericalEditInRange** can be implemented as a *text_field*, a *spin_box* or a *track_bar*
- A **PositionEdit** can be implemented as an *image_map*
- A **SingleChoice** can be implemented as a *radio_button*, *list_box*, *drop_down_list* or *image_map*
- A **TextEdit** can be implemented as a *text_field* or a *text_area*.

Concrete Vocal User Interface

While in graphical interfaces the concept of presentation can be easily defined as a set of user interface elements perceivable at a given time (e.g. a page in the Web context), in the case of vocal interfaces we consider a presentation as a set of communications between the vocal device and the user that can be considered as a logical unit, e.g. a dialogue supporting the collection of information regarding a user. The AUI refinements for obtaining the Vocal CUI definition involves defining some elements that enable setting some presentation properties. In particular, we can define the default properties of the synthesized voice (e.g. volume, tone), the speech recognizer

(e.g. sensitivity, accuracy level) and the DTMF (Dual-Tone Multi-Frequency) recognizer (e.g. terminating DTMF char).

The following are the interactors refinements:

- An **Alarm** can be implemented as a pre-recorded *sound*
- A **Description** can be implemented as:
 - *speech*, which defines text that the vocal platform must synthesize or the path where the platform can find the text resources. It is furthermore possible to set a number of voice properties, such as emphasis, pitch, rate, and volume as well as age and gender of the synthesized voice. Moreover, we have introduced control of behaviour in the event of unexpected user input: by suitably setting the element named *barge in*, we can decide if the user can stop the synthesis or if the application should ignore the event and continue.
 - *pre-recorded message*, which defines a pre-recorded audio resource, with an associate alternative content in case of unavailability.
- A **MultipleChoice** can be implemented *vocal selection*. This element defines the question(s) to direct to the user and the set of possible user input that the platform can accept. In particular, it is possible to define textual input (word or sentences) or DTMF input. In this version, the interactor accepts more than one choice
- A **SingleChoice** can be implemented as a *vocal selection* that accepts only one choice.
- An **Activator** can be implemented as a *command*, in order to execute a script, a *submit*, to send a set of data to a server, and *goto* to perform a call to a script that triggers an immediate redirection
- A **Navigator** can be implemented as a *goto* for automatic change of presentation, a *link* for user-triggered change of presentation, and a *menu* for supporting the possibility of multiple target presentations.
- A **TextEdit** can be implemented as *vocal textual input* element, which permits setting a vocal request and specifying the path of an external grammar for the platform recognition of the user input.
- A **NumericalEditFull** and **NumericalEditInRange** can be implemented as a *vocal numerical input*, which accepts only numbers (in a range in the latter case) specified through a grammar.
- An **ObjectEdit** can be implemented as a *record* element, which allows specifying a request and storing the user input as an audio resources. It is possible to define a number of attributes relative to the recording, such as *beep* to emit a sound just before recording, *maxtime* to set the maximum duration of the recording, and *finalsilence*, to set the interval of silence that indicates the end of vocal input. Record elements can be used for example when the user input cannot be recognised by a grammar (e.g. a sound).

With respect to the composition of interactors, the Vocal CUI has four solutions that permits to identify the beginning and the end of grouping:

- Inserting a sound at the beginning and at the end of the group

- Inserting a pause , which must be neither too short (useless) nor too long (slow system feedback)
- Change the synthesis properties (such as volume and gender)
- insert keywords that explicitly define the start and the end of the grouping

Another substantial difference of vocal interfaces is in the event model. While in the case of graphical interfaces the events are related mainly to mouse and keyboard activities, in vocal interfaces we have to consider different types of events: noinput (the user has to enter a vocal input but nothing is provided within a defined amount of time), nomatch, the input provided does not match any possible acceptable input, and help, when the user asks for support (in any platform specific way) in order to continue the session. All of them have two attributes: message, indicating what message should be rendered when the event occurs, and re-prompt, to indicate whether or not to synthesize the last communication again.

[Paterno2000] F. Paternò, C. Santoro, L.D. Spano, "MARIA: A Universal Language for Service-Oriented Applications in Ubiquitous Environment", ACM Transactions on Computer-Human Interaction, Vol.16, N.4, November 2009, pp.19:1-19:30, ACM Press.

4.5 MBUI WG Note - Introduction to Model-Based UI Design

This document is currently in preparation and is expected to be published as a W3C Working Group Note in September 2013. The document provides introductory material describing model-based user interface design, its benefits and limitations, and a range of illustrative use cases.

- [Editor's Draft](#)

4.6 MBUI WG Note - Car Rental Use Case

At the Munich face to face, the Working Group decided to split off the description of the Rental Car use case into a separate document that could be referenced from the introduction to model-based UI, and the two meta-model specifications. The use case focuses on the user interface for booking a rental car, and we have been able to base this on real-world examples such as the Hertz Rental Car agency's website and Android application for smart phones.

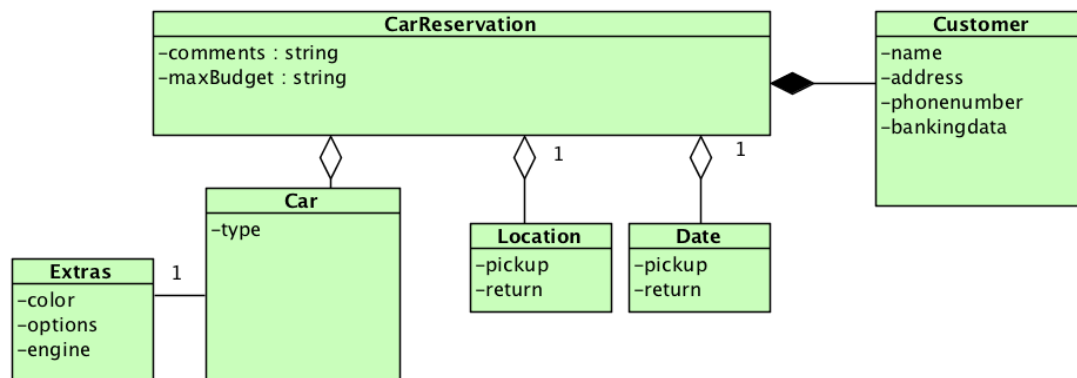
- [Editor's Draft](#)

4.6.1 Introduction

The car rental example consists in a scenario in which the interactive system supports users in the task of renting a car. In this sense, various context information can be used to adapt application aspects, and to properly display the list of cars to rent, enabling users to make choices and to accomplish the main task.

Basically, users must provide information about the car (i.e. category, color, model, and engine), then their own information (i.e. name, surname, address, city, ZIP code, country, gender, birthday and email), and finally other information (i.e. commentaries, and maximum budget). Once the preferences are set and the request is submitted, users access the service and the results. To conclude the rental, users select a car, and define the period of interest. This task model is merely illustrative, and serves as a basis for the implementations, mainly because some of the CAAs envisaged affect the tasks' sequence, so specializations of this model are expected.

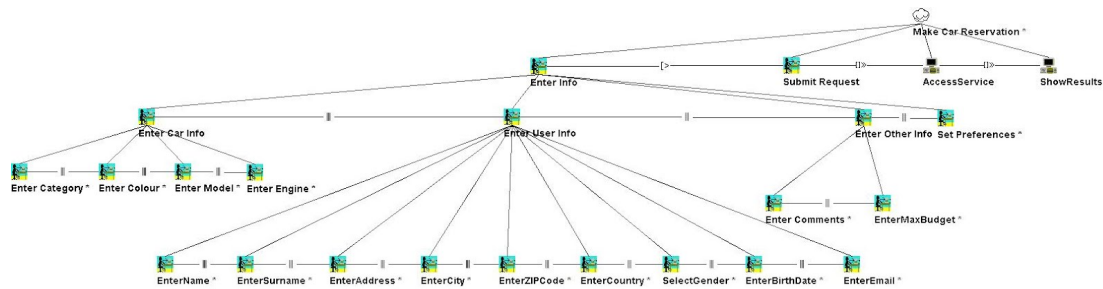
The domain model specifies a basis for the data structure of the form fields, and defines which of the information required is mandatory or optional. This model serves as guidance for the implementation examples.



4.6.2 Task Model

A set of key functional requirements must be considered for implementing the car rental example. The users must be able to:

- select the city of interest to pick up the car;
- specify the period for the car rental;
- access a set of possible cars and select one;
- to see details about the car of interest;
- access and select additional car features (e.g. GPS);
- provide personal information before renting the car;
- access details about the car rental before submitting the request;
- change the car rental parameters anytime before confirming the rental.



4.7 MBUI WG Note - Glossary of Terms

This document is currently in preparation and is expected to be published as a W3C Working Group Note in September 2013. The document provides definitions for a range of terms used for model-based user interface design, and is targeted at would be adopters of model-based user interface design techniques. In working on this document, we have noticed that different practitioners of model-based user interface design techniques often use slightly different terminology, and moreover, there is an understandable tendency for this to be focused on the needs of academic study as opposed to that of industrial users. We have therefore taken a selective approach to which terms we are including in the glossary.

- [Editor's Draft](#)

4.8 MBUI WG Specification - Task Models for Model-Based UI Design

- <http://www.w3.org/TR/2012/WD-task-models-20120802/>

This is a specification document that the Model-Based User Interfaces Working Group is progressing along the W3C Recommendation Track with a view to attaining Recommendation status by the end of the current charter period (November 2013). The First Public Working Draft was published on 2nd August 2012. An updated public working draft is expected in August 2013.

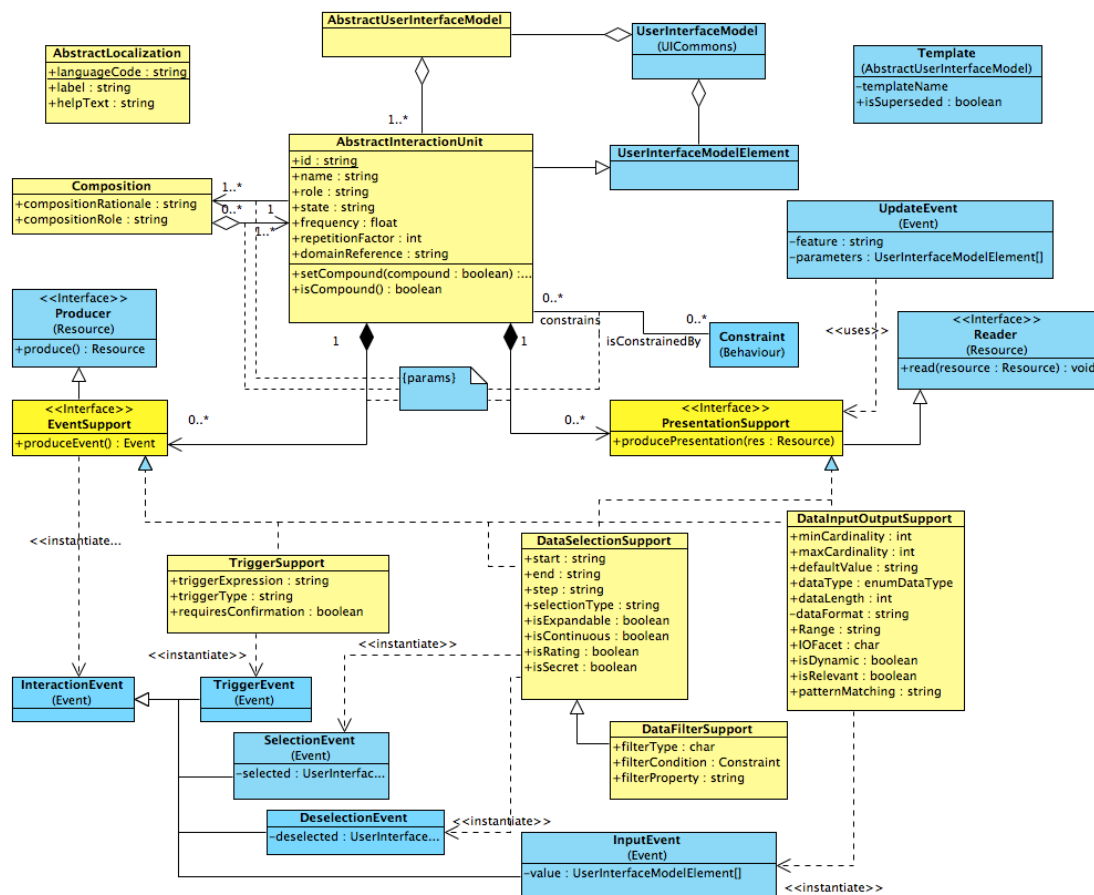
The specification is based upon the ConcurTaskTree (CTT) notation and refines the metamodel introduced in earlier versions of CTT.

4.9 MBUI WG Specification - Abstract User Interface Models

- <http://www.w3.org/TR/2013/WD-abstract-ui-20130926/>

This document is expected to be published as a W3C Working Draft in late September 2013 at the above URL. The document specifies a metamodel and interchange format for abstract user interface models. This took longer than originally envisaged in the Working Group Charter due to the need to assimilate ideas from all of the various Working Group submissions, and to reach a broad consensus on a merged approach.

The following diagram presents the metamodel as of the beginning of August 2013 and can be compared with the metamodel for ASFE-DL:



4.10 Relationship to IndieUI Working Group

The W3C IndieUI Working Group is developing specifications covering:

- Device independent user interaction events such as scrolling or changing values
- Providing Web applications with limited access to the user needs and preferences

The idea of there being events at different levels of abstraction fits nicely into the Cameleon Reference Framework, and the ability to specify behavior cleanly at each level of abstraction. The work item on user needs and preferences relates to the Cameleon Reference Framework's context of use. The IndieUI Working Group was motivated by the desire for improvements in the accessibility of Web applications, and follows on from work on WAI-ARIA (Accessible Rich Internet Applications). Developers are increasingly creating custom user interaction controls with Web page scripts. WAI-ARIA defines techniques for annotating such controls with properties that browsers can exploit to identify the role and state of custom controls for driving assistive technology such as braille readers.

- [IndieUI Working Group](#)
- [IndieUI Events](#)
- [editor's draft](#)
- [WAI-ARIA Overview](#)

4.11 MBUI WG Future Plans

The W3C Model-Based User Interfaces Working Group is currently chartered until November 2013. During this period, we are attempting to standardize metamodels and interchange formats for task models and abstract user interface models. We are also working on supplementary information covering the rationale for adopting model-based user interface design techniques, exemplary use cases, and a glossary of terms.

Progress has been considerably slower than initially anticipated, and it has taken over two years to refine the editor's draft for the meta model for the abstract user interfaces to the point where it can be released as a W3C First Public Working Draft. This time has been used for reviewing different modeling approaches and building a consensus around a new synthesis of ideas. A revised public working draft for the meta model for tasks has been published with relatively minor changes.

The current plan is to move both specifications to Last Call Working Drafts and seek external review. We would then aim to transition the specifications to W3C Candidate Recommendations together with calls for implementations. The Model-Based UI Working Group would then be put into a dormant state, awaiting either a charter extension to deal with feedback arising from the implementation reports, or rechartering with a new scope and set of work items.

In the meantime, we have launched the W3C Ubiquitous Application Design Community Group to foster dialogue between developers and researchers, including use cases, requirements, experience with existing tools and plans for future standards. The Community Group will collect feedback on the Model-Based UI working group specifications and associated implementation reports, and make recommendations for further work.

Further opportunities for standardization could include:

- metamodels and interchange formats for the context of use

- rule languages for mappings between layers in the CAMELEON Reference Framework and for adaptation to the context of use at both design time and run-time
- metamodels and interchange formats for the Concrete User Interface

5 W3C Ubiquitous Application Design Community Group

W3C Community Groups are free of charge and intended to provide a forum for developers, designers, and anyone passionate about the Web to have discussions and publish documents. This includes developing specifications, holding discussions, developing test suites, and connecting with W3C's international community of Web experts.

- <http://www.w3.org/community/>

The W3C Ubiquitous Applications Design Community Group seeks to bring together developers and researchers to explore and promote techniques for context aware design that separates out different aspects of design to speed development and reduce costs. The Community Group plans to do this through gathering and discussing techniques, together with developing open source demonstrators. The Group's website can be found at:

- <http://www.w3.org/community/uad/>

At the time of writing, the Group has only recently formed and is at an early stage where we are recruiting members and starting to put content into the wiki. We have attracted people from within the Serenoa project and the W3C Model-Based UI Working Group, and now need to find ways to market the Group to developers. A promising way forward would be to conduct an open survey of developer challenges and available tools for designing applications that can be used effectively across desktop, smart phones, tablets and connected TVs.

Web developers are already aware of what is known as **Responsive Design**. This is a collection of techniques for cross platform design, with a particular emphasis on image resources. Loading a high resolution image on a device with a small display, increases the load time, wastes device memory and battery life, and may cost the user more money if they are not on an unlimited data plan. On the other hand, loading a low resolution image on a device with a large display will result in small or blurry images, spoiling the user experience. The advent of double density displays (e.g. iPad 2 with its Retina display) only exacerbates the problem.

Adaptation of web pages can be done server side based upon information on the device carried as part of HTTP requests. This typically involves having to consult a database of device characteristics. Client-side adaptation can be addressed through a combination of CSS media queries, image srcset definitions, and JavaScript APIs. CSS media queries provides a means to adjust the web page styling according to the

device characteristics, especially the display details. The srcset attribute allows designers to specify alternative versions of an image resource that are appropriate for different display resolutions. Web page scripts can also use JavaScript APIs to sense and adapt to the device characteristics, including the display resolution, its orientation, the ambient light level and so forth.

Although techniques for responsive design are now available, few companies are as yet exploiting them, with surveys showing that many companies design their websites for desktop users, and offer a relative poor user experience on mobile browsers. Instead, users may be able to install native applications, e.g. from Apple's app store for iPhone and iPad, and for Android phones and tablets from Google's Play store. There is also a lack of effective authoring tools for developing cross platform web applications, and this represents a significant market opportunity. The Ubiquitous Applications Design Community Group needs to involve developers in gathering information on existing tools and techniques and identifying ideas for the kinds of tools that are needed. Ideally, it would be possible to stimulate a community effort on open source tools as a means to demonstrate what is possible.

One challenge is in attracting large companies, who are concerned by the difficulties of performing a review of patent commitments before joining community groups. This problem only arises for W3C Community Groups that indicate a desire to propose specifications for future standardization. The solution will be for the Ubiquitous Applications Design Community Group to propose and adopt a formal charter for work on specifications. The drafting of the charter will have to wait until the Group has first clarified the use cases and requirements.

For the Community Group to be successful, we will need individuals who are willing to carry out work through email and editing the wiki, but also through outreach to external groups, such as web developer forums. This represents a significant hurdle for the staff involved in the Serenoa project when the funding ceases at the end of September 2013, and the people involved are redirected to other work.

A W3C colleague suggested the following approach:

1. Write up very clearly what you want people to comment on/give opinion on, and why this is useful to developers, and by extension, users. Perhaps in 4-5 short bullet points. the main page you link to below has good info, but it is still a bit long, and people may well just go "TL;DR"
2. Provide a really easy feedback mechanism. A wiki is ok, but a lot of people hate using wikis. Some targeted questions on a short google form/survey might be more effective.
3. Write an interesting discussion piece discussing the topic, its issues, potential solutions going forward, and a call to action, which can include a link to the survey/wiki/however we decide to do it. I can put you in touch with contacts from A List Apart and dotnet mag, to get that published and out to a wider audience. I think IanJ already has ALA in hand, with that regular column W3 are now doing, but I'm not sure if he has dotnet mag contacts.
4. Talk about it at conferences, like W3 conf, others?

5. Draw up a list of the most influential people in this area, whose opinions would be really useful, and who could spread the word far and wide. Get W3 to add to the list.
6. Once you've got a list of say 25, ask them to retweet and blog about it.

He was kind enough to provide a list of key contacts for initial outreach (not included here for privacy reasons).

The plan is to draft a survey form and send it out in September 2013 in conjunction with the outreach suggested above. Hopefully this will encourage people to join the Community Group, and to find volunteers willing to act as Chairs.

6 CoDeMoDIS proposal for a COST Action

COST (European Cooperation in Science and Technology) is a long running intergovernmental framework supporting cooperation among scientists and researchers across Europe.

- <http://www.cost.eu/>

A proposal for a COST Action was prepared to support collaboration on Model Driven Engineering (MDE) and Model-Based User Interface Development (MBUID). If approved, this would have fostered continued collaboration beyond the end of the Serenoa project, and provide an opportunity for supporting involvement in further work on standardization in addition to work on harmonization between the currently distinct fields of MDE and MBUID. The proposal plans to set up working groups on the following topics:

- Taxonomy of Model-Driven Engineering of Interactive Systems
- Comparative Analysis of Models, Methods, and Related Technologies
- Software Support for Model-Driven Engineering of Interactive Systems
- Harmonization and Unification of Standardisation Efforts

In addition, a Standardization Coordinator would be assigned in order to coordinate all efforts towards standardization.

The participants behind the proposal come from a broad range of countries, including Austria, Belgium, Bulgaria, Switzerland, Cyprus, Czech Republic, Germany, Denmark, Estonia, Greece, Spain, Finland, France, Croatia, Hungary, Iceland, Ireland, Israel, Italy, Luxembourg, Macedonia, Norway, Poland, Portugal, Romania, Serbia, Sweden, Slovenia, Slovakia, United Kingdom, Argentina, Japan, New Zealand, and the United States. The proposer is Dr. Gerrit Meixner, German Research Center for Artificial Intelligence (DFKI), Kaiserslautern, Germany.

The pre-proposal was accepted, and the full proposal submitted for review at the end of July 2012.

7 ISO 24744 standardisation action

ISO/IEC 24744 Software Engineering — Metamodel for Development Methodologies is an international standard focusing on the use of meta models for software development methodologies for information-based domains.

- http://www.iso.org/iso/catalogue_detail.htm?csnumber=38854

A standardization action has been suggested to harmonize the ISO 24744 methodologies with Model-Based User Interface Development techniques. At this point, this is very much in the early planning stage.

8 Conclusions

This report surveys the standardization prospects for ideas emerging from the Serenoa project and describes the progress made in the W3C Model-Based User Interfaces Working Group. A First Public Working Draft has been published for task models and will soon be followed by another for abstract user interface models, based upon a synthesis of ideas from a range of submissions to the Working Group. The aim is to progress these to W3C Recommendations by the time the Working Group's Charter draws to an end in November 2013. That milestone is now looking unlikely and we are instead hoping to produce W3C Candidate Recommendations, and for the Working Group to then be put on hold until implementation reports are available along with the resources to re-charter the MBUI Working Group. The new W3C Ubiquitous Application Design Community Group is intended as a forum to bring developers and researchers together, and to seek implementation experience with the specifications from the MBUI Working Group. The Community Group is free to join, and open to all. It will be able to collect ideas for chartering further standards work on Model Based UI design at W3C.

A major challenge will be to convince industry of the practical value of model-based user interface design techniques, and this will require investment in developing robust tools and run-time environments as well as outreach on the business case for adoption. The Serenoa project is playing a key role in supporting this work, but further investment will be needed beyond the end of the project if Europe is to realize the opportunities for exploiting model-based user interface design techniques. This is all the more important given the current trend towards a wider range of user interface technologies and device platforms. Further work should also take into account the emergence of the Internet of Things as a driver for new kinds of user interfaces, along with the importance of multilingual user interfaces to support interaction in people's native languages.

9 References

D2.1.2

CARF and CADS http://www.serenoa-fp7.eu/wp-content/uploads/2012/07/SERENOA_D2.1.2.pdf.

D2.3.2

CARFO and CARFO Population http://www.serenoa-fp7.eu/wp-content/uploads/2013/02/SERENOA_D2.3.2.pdf.

D3.3.2

AAL-DL http://www.serenoa-fp7.eu/wp-content/uploads/2013/10/SERENOA_D3.3.2.pdf.

D3.2.3

ASFE-DL http://www.serenoa-fp7.eu/wp-content/uploads/2013/10/SERENOA_D3.2.3.pdf.

D5.2.3

Application Prototypes http://www.serenoa-fp7.eu/wp-content/uploads/2013/10/SERENOA_D5.2.3.pdf.

AIM

<http://www.multi-access.de/mint/aim/2012/20120516/>.

AutoBG

W3C's Automotive and Web Platform Business Group <http://www.w3.org/community/autowebplatform/>.

Balsamiq

Balsamiq Mockups rapid wireframing tool <http://balsamiq.com/>.

BREW

Binary Runtime Environment for Wireless (BREW), <https://www.brewmp.com/>.

CAMELEON: Calvary2002

The CAMELEON Reference Framework, G. Calvary, J. Coutaz, D. Thevenin, L. Bouillon, M. Florins, Q. Limbourg, N. Souchon, J. Vanderdonckt, L. Marucci, F. Paternò, and C. Santoro, CAMELEON Project, September 2002.

CTT: Paterno:1997

Paternò, F., Mancini, C. and Meniconi, S., ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models, Proceeding INTERACT '97 Proceedings of the IFIP TC13 International Conference on Human-Computer Interaction, Pages 362-369.

ECMA

ECMA International, <http://www.ecma-international.org/>.

ETSI

<http://www.etsi.org/>.

Kumar1992

Kumar, V., Algorithms for constraint satisfaction problems. AI Magazine, 1992.
<http://www.cs.cinvestav.mx/~constraint/papers/kumar-survey.pdf>.

J2ME

Java Platform, Micro Edition, see http://www.java.com/en/download/faq/whatis_j2me.xml.

Justinmind

Wireframe tool for defining mobile and web apps <http://www.justinmind.com/>.

Maquetta

Visual HTML5 authoring tool <http://maquetta.org/>.

MARIA: Paterno2009

Paternò, F., Santoro, C. and Spano, L. D. *MARIA: A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments*, ACM Trans. Comput.-Hum. Interact., ACM, 2009, 16, 19:1-19:30.

MBUI Working Group Charter

<http://www.w3.org/2011/01/mbui-wg-charter>.

Motti2013

UsiXML website: www.usixml.org

Motti, V., Raggett, D., Cauwelaert, S. and Vanderdonckt, J., Simplifying the Development of Cross-Platform Web User Interfaces by Collaborative Model-based Design, in Proc. SIGDOC 2013.

W3C Process

<http://www.w3.org/Consortium/Process/>.

MXML

<https://learn.adobe.com/wiki/display/Flex/MXML>

OASIS

<https://www.oasis-open.org/>.

OMG

<http://www.omg.org/>.

OpenLaszlo

<http://www.openlaszlo.org/>

Petrie1986

Petrie C.J., "New Algorithms for Dependency-Directed Backtracking", Technical Report AI TR86-33, Austin, Texas: Artificial Intelligence Laboratory, The University of Texas at Austin, 1986.

<ftp://ftp.cs.utexas.edu/pub/AI-Lab/tech-reports/UT-AI-TR-86-33.pdf>.

PhoneGap

<http://phonegap.com/>.

QVT

Query/View/Transformation v1.1 <http://www.omg.org/spec/QVT/1.1/>.

Raggett2013

Motti, V. and Raggett, D. *Quill: A Collaborative Design Assistant for Cross Platform Web Application User Interfaces*, WWW 2013, Developer's Track, <http://www2013.org/companion/p3.pdf>.

RETE

Charles Forgy, "Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem", *Artificial Intelligence*, 19, pp 17–37, 1982.

RIF

Rule Interchange Format <http://www.w3.org/TR/rif-overview/>.

SketchFlow

Prototyping tool that is part of Microsoft Expression Studio

Ultimate <http://www.microsoft.com/expression/eng/>, see

[http://msdn.microsoft.com/en-us/library/ee341449\(v=expression.40\).aspx](http://msdn.microsoft.com/en-us/library/ee341449(v=expression.40).aspx).

SerenoaWP

Serenoa White Paper http://www.serenoa-fp7.eu/wp-content/uploads/2010/10/white_paper_v1.0.pdf.

Souchon2003

Souchon, N. and Vanderdonckt, J. *A Review of XML-Compliant User Interface Description Languages*, Proc. of 10th Int. Conf. on Design, Specification, and Verification of Interactive Systems DSV-IS'2003 (Madeira, 4-6 June 2003), Jorge, J., Nunes, N.J., Falcao e Cunha, J. (Eds.), Lecture Notes in Computer Science, Vol. 2844, Springer-Verlag, Berlin, 2003, pp. 377-391.

UIML: Abrams1999

Abrams, M., Phanouriou, C., Batongbacal, A.L., Williams, S. & Shuster, J. (1999), UIML: An Appliance-Independent XML User Interface Language. In A. Mendelzon, editor, Proceedings of 8th International World-Wide Web

Conference WWW'8 (Toronto, May 11-14, 1999), Amsterdam, 1999. Elsevier Science Publishers. See <http://www8.org/w8-papers/5b-hypertext-media/uiml/uiml.html>.

Ueware: Zuehlke2008

Zuehlke, D.; Thiels, N.: Ueware engineering: a methodology for the development of user-friendly interfaces, in: Library Hi Tech, Vol. 26, No. 1, 2008.

UsiXML: Vanderdonckt2004

UsiXML website: www.usixml.org

Vanderdonckt, J., Limbourg, Q., Michotte, B., Bouillon, L., Trevisan, D., Florins, M., UsiXML: a User Interface Description Language for Specifying Multimodal User Interfaces, in Proc. of W3C Workshop on Multimodal Interaction WMI'2004 (Sophia Antipolis, 19-20 July 2004).

W3C

World Wide Web Consortium, <http://www.w3.org/>.

W3C Process

<http://www.w3.org/Consortium/Process/>.

XAML

Microsoft's XAML, <http://msdn.microsoft.com/en-us/library/ms752059.aspx>

XForms: Boyer2009

J. Boyer. W3C Recommendation, 20 October 2009. Available at: <http://www.w3.org/TR/2009/REC-xforms-20091020/>. Latest version: <http://www.w3.org/TR/xforms11/>.

XSLT

<http://www.w3.org/TR/xslt>.

XUL

XML User Interface Language, <http://developer.mozilla.org/en/XUL>