CASSIS: A Modeling Language for Customizable User Interface Designs

CASSIS: a Modeling Language for Customizable User Interface Designs

Jan Van den Bergh and Karin Coninx

Hasselt University - tUL - iMinds
Expertise Centre for Digital Media
Wetenschapspark 2, 3590 Diepenbeek, Belgium
Email: firstname.lastname@uhasselt.be

Abstract. Current user interface modeling languages usually focus on modeling a single user interface and have a fixed set of user interface components; adding another user interface component requires an extension of the language.
In this paper we present CASSIS, a concise language that supports creation of user interface components using models instead of language extensions. It also allows the specification of design-time and runtime user interface variations. The support for variations has been used to generate constraints for custom user interface components, to specify design patterns and design decisions. CASSIS has been used in several projects including a multi-disciplinary applied research project. [1]

1 Introduction and Related Work

This paper introduces CASSIS, Context-Aware SyStem Interaction Specification, a tool supported language to model the interaction between humans, computing systems and their user interfaces. CASSIS contains generic constructs with a corresponding (graphical) notation to model user interfaces, events, related concepts and their interactions. The language has specific constructs that allow flexible, customizable reuse of models or parts thereof through the definition of custom types and templates or interaction patterns that can be bundled in a library. These libraries can later be imported in new projects.

CASSIS supports the specification of design options, design decisions (accept, reject, runtime option), which can be used to define the structure and behavior of the specified user interface and as well as the usage context of custom types.

Several languages have been defined that target modality-independent specification of user interfaces. Van den Bergh et al. [1] give a reasonable overview of several of these languages. In contrast to CASSIS, the languages in their overview all contain a fixed amount of user interface components. Their proposal, CAP3, defines four user interface components as part of the language and defines other user interface components in a library. They, however, do not specify how this functionality is realized. In contrast to CASSIS, CAP3 is completely focused

[1] The original publication is available at www.springerlink.com.

on Canonical Abstract Prototypes and extensions thereof. It does not include events, nor design options and has no possibility to define the structure of new components.

IFML beta 1[2] is a more recent proposal for an Object Management Group standard. It does support graphical specification of events, but has a fixed set of user interface components and no support for context-awareness. A preview of the final IFML standard [2] promises to include modules that allow to define reusable flows, but the concept differs from user interface components. In contrast to CASSIS, IFML exclusively focuses on interaction flow and cannot be used to specify user interface structure.

UsiXML [7,15] consists of several submodels, including a modality-independent specification of user interfaces, a context model, a domain model and several other user interface related models. UsiXML has a fixed number of user interface components.

MARIA [11] is an XML-based language that allows specification of user interfaces at different levels of abstraction. Similar to UsiXML, MARIA supports a fixed number of user interface components and its dialog expressions are connected using CTT [8] operators and it can target multiple platforms.

CUP 2.0 [14] is an UML-profile [9] that allows the specification of context-aware user interfaces. It has a fixed number of user interface components. It defines the user interface structure and behavior using two different diagrams (class diagram and activity diagram). Context-awareness is supported within the activity diagram.

UIML [4] is an XML-based language that allows the definition of user interfaces based on custom user interface component vocabularies, which are linked to very generic parts. In contrast to CASSIS, parts do not contain any specific information regarding the properties of the user interface component. All properties are specific for a UIML vocabulary. Some implementations provide support for multiple platforms.

Design decisions, visual layout similar to concrete layout (both requested by UX practitioners in interviews)were not addressed in (these) model-based approaches at this level of abstraction. Similarly visual specification of custom user interface components and patterns with custom icons including constraint generation, which allow the user to adapt the notation (a bit) to their specific domain, were not addressed.

2 CASSIS

CASSIS is a modeling language that is defined by a metamodel (a model that defines the structure of all CASSIS models), constraints and its concrete syntax.

This section introduces the metamodel as well as the graphical concrete syntax. This graphical syntax is not completely defined within the language itself. CASSIS supports the creation of libraries of components, concepts and events, which can specify their own icon. Constraints will be informally discussed.

[2] `http://www.slideshare.net/mbrambil/ifml-omgftfreportsanta-claracadec2013`

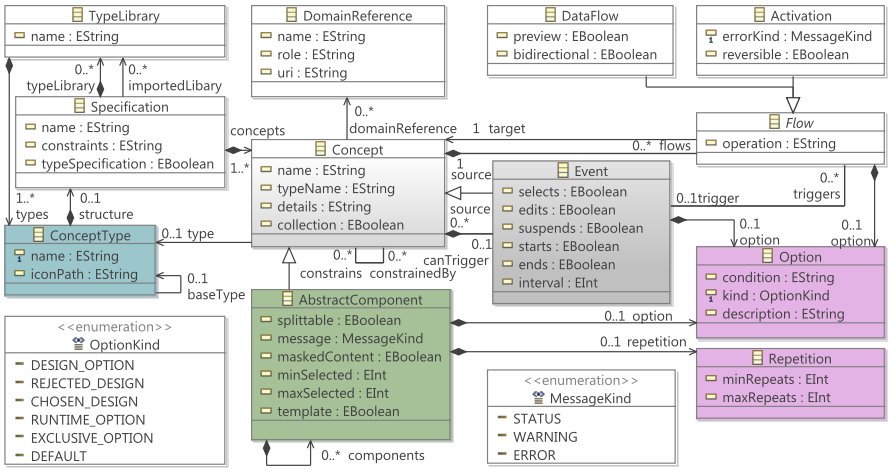


Fig. 1. The complete CASSIS metamodel

2.1 Metamodel

The CASSIS metamodel (Fig. 1) is defined using ECore [13, Chapter 5], which is part of the Eclipse Modeling Framework (EMF). It consists of twelf metaclasses (and two enumerations):

- *Concept*, *Event* and *AbstractComponent* define the core constructs of the language,
- *TypeLibrary*, *ConceptType* and *Specification* are used to enable the definition of system interaction, custom types, templates and patterns,
- *Flow*, *DataFlow* and *Activation* define the control and information flow,
- *Repetition* defines how many times an *AbstractComponent* can be present in a specific part of a user interface,
- *Option* defines conditional presence of *Event*s and *AbstractComponent*s, and,
- *DomainReference* allows to specify relations between the core constructs and constructs in other resources (models or other files).

?	DESIGN_OPTION		RUNTIME_OPTION
✓	CHOSEN_DESIGN		DEFAULT
✗	REJECTED_DESIGN		EXCLUSIVE_OPTION

Table 1. Possible option types and corresponding icons

2.2 Behavior of Interactive System

The behavior expressed by a CASSIS model is determined by events, flows and option-objects and how they connect to abstract components and concepts. For this discussion of a system's behavior specification by CASSIS, we consider only two states of a concept; active and inactive. Active components are components that are available to a user to interact with. This does not mean that for graphical user interfaces the component is visible on a screen. It means that a user can start interacting with it without having to first change the state of the functional core. E.g. changing a tab in a Ribbon interface or a tabbed dialog does not imply a change to the functional core. The meaning of active or inactive is undefined for a Concept in general. It can be defined on a case by case basis. E.g. for a software component it could be defined that there exists at least one instance, for a user role, it could mean that a user with this role is logged in or for a geographical location it could mean that its GPS location is recorded in a specific database.

Abstract components that are contained by the same component (or by no component at all) by default execute in parallel; users can interact with these components in any order and without having to complete interaction with any of them at any time. Users can even interact with several components at the same time. This corresponds to how users can interact with user interface components (widgets) contained in the same container (a window) when developers do not specify constraints between these components. This ensures that this part of the behavior coincides with what a developer might expect when using the visual syntax of CASSIS.

When one or more of the abstract components have an *Option* attached to them, this may impact this behavior; a *DEFAULT* option indicates the component that users can interact with first; it gets the default focus. There should only be one *DEFAULT* component within an abstract component. In Fig. 3, *Main Window* is the *DEFAULT* component. A *RUNTIME_OPTION* indicates that availability of the component is dependent on constraints that can be changed at runtime. Only one of all components with an *EXCLUSIVE_OPTION Option* can be active at any time. Which one this is, is determined by associated constraints. The remaining values for options have no impact on the behavior of the specified system other than static specification of the availability of components.

Three event attributes specify an effect on the state of the event's source when they occur:

1. *starts*: the event never deactivates the source. When an activation relation connects the event with a *target* component, it starts in parallel.
2. *suspends*: when the source is connected through an *Activation* with a *target* component, the source is suspended until the corresponding activity *ends*. This may mean that other abstract components are activated before the originating abstract component is reactivated. E.g. A may be the first screen of a wizard while the originating component is only reactivated after the whole wizard is completed (or canceled). When the event is not connected to an *Activation* the behavior is unspecified.
3. *ends*: the source of the event becomes inactive in all cases.

When none of the above attributes is true the source is only deactivated when the event is connected to an *Activation* whose *target* component does not have an ancestor in common with the originating component. We believe that the behavior that can be expressed using these event attributes and *Option* and *Repetition* is at least as powerful as using CTT operators as defined in [8].

2.3 Type Specification

A user can use a *ConceptType* to customize the meaning of a concept using a type *Specification*, which can be expressed in a dedicated diagram, as well as its appearance with a user-defined icon. To specify constraints on the application of the type to concepts, one should define the *structure* using a type specification. A type specification can contain manually specified *constraints*. Constraints can also be generated based on the *ConceptType*'s *structure*.

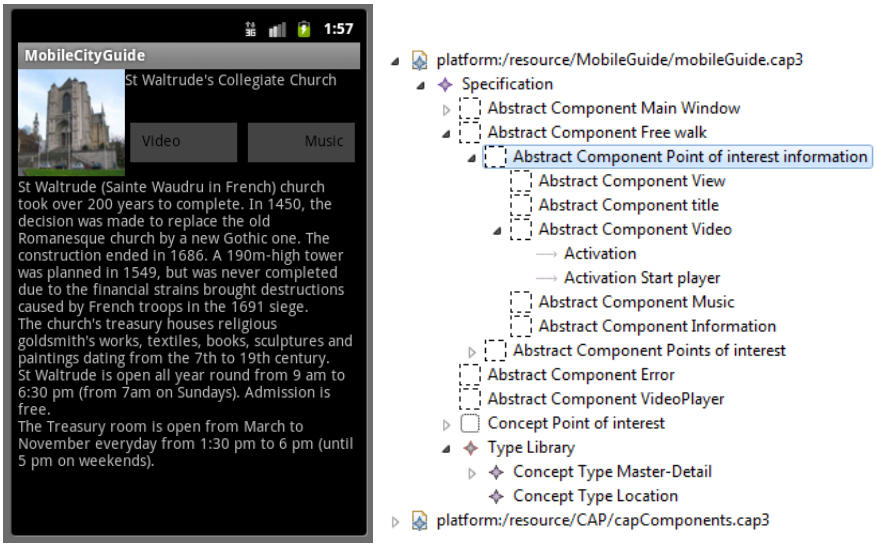
For all concepts the attribute values and associations are translated in errors or warnings. Using EGL [12] we generate EVL [5] to propose automated fixes for these constraints. Constraints for the inner structure are only generated for *template* components. When the template attribute of a component is false, its inner structure is considered as human documentation rather than constraints that should be validated by a tool.

Although all flows should originate from *Event*s, it is not necessary to always specify these Events if a concept type is applied to a *Concept* or *AbstractComponent*; When the event can be uniquely identified based on the type specification, the event does not have to be repeated on a *Concept* with this type specification. This significantly reduces the number of elements in a model (see Fig 3).

2.4 Mobile City Guide

We use the mobile city guide presented by Degrandsart et al. [3] as a *specification exemplar* of a context-aware application. We present only the parts of the exemplar relevant for discussing the features of CASSIS.

The mobile city guide is an interactive application running on mobile devices (such as a smart phone or a tablet PC) that presents information about points of interest in the vicinity of the person carrying the device. The guide has two modes: a guided mode and a free walking mode. The free walking mode is considered to be more important for the users and only this mode will be implemented for the first version of the mobile guide (as indicated by the DEFAULT and REJECTED *Option*s indicating design-time variations). To increase interactivity, the information is displayed when the user is approaching one of these points of interest. The presented information can be of various kinds: pictures, text, video, sound,... When approaching a point of interest, the application should quasi-instantaneously present a picture, a name and a small description and allow a user to request more information if necessary. Video and sound may be played on demand, depending on the availability of a connection to an external database and memory available on the device. The mobile city guide depends on network availability to play this video.



(a) High-fidelity prototype (b) Model of the guide

Fig. 2. Mobile city guide: (a) screenshot of a prototype the Point of interest information, and the corresponding part of the CASSIS model highlighted in (b)

Fig. 3 shows the graphical CASSIS specification of the mobile city guide. Fig. 2(b) shows the corresponding CASSIS model using a reflective EMF editor in Eclipse. The usage of the library of Canonical Abstract Prototypes components is shown on the last visible line in Fig. 2(b).

The mobile guide has three main parts: the *Main Window* (the start screen, as indicated by the *DEFAULT* option), an overview of all points of interest (*Free Walk* and details about a Point of Interest *Point of interest information*. A screenshot of the latter in a high-fidelity prototype is shown in Fig. 2(a). This screen is automatically shown when the user of the mobile guide is near a point of interest (*Activation* from *near Point of interest* to *Point of interest information*). The automated display of information is however dependent on user-configurable settings (*RUNTIME_OPTION*) at runtime. From this screen, the user can activate a video. As this video is downloaded over the network, this may cause an error when no network is available, modeled through a thick red *Activation* arrow to the *Error* message.

3 Assessment and discussion

At several stages during its development CASSIS was discussed with interaction designers and user interface developers from industry and academia. These discussions formed the basis for major and minor revisions to CASSIS. Making these changes was greatly eased through tool support, which was created and frequently recreated using Eclipse EMF [13] and Epsilon [6].

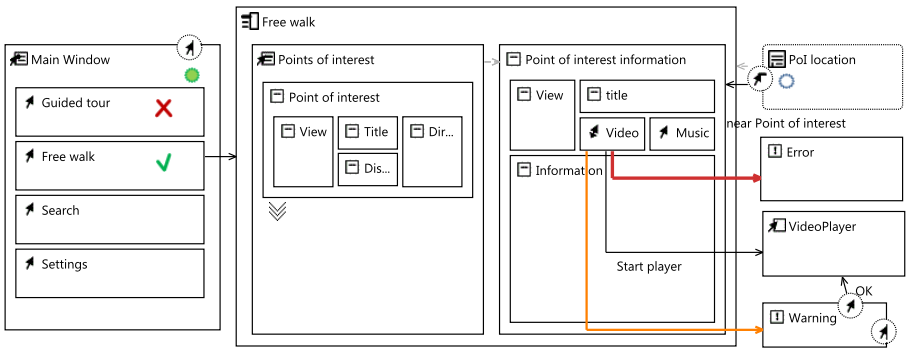


Fig. 3. Mobile City Guide specification using CASSIS; the CAP library; *Free walk*, a master-detail template component; and a custom Location concept (Fig. 2(b), bottom)

The developed models had varying sizes. The largest model had over 100 model elements (excluding the Canonical Abstract Prototypes library used to create the model). It was successfully used to discuss the structure and behavior of a context-aware mobile application, targeting professional caregivers for people with dementia, with academic and industrial partners in an applied research project[3]. The people involved in the discussion included 4 persons with a software development and software engineering background, a cognitive scientist and a designer. The fact that the screens were presented within their interaction context, meant that the discussion was not only focused on the structure, but also on the behavior of the designed application. The ability of the supporting tool to hide the details of structures nested within a component was also helpful to focus the discussion on a particular part of the interface.

Proof-of-concept tool support for CASSIS has been realized using Eclipse EMF [13], and a subset of the Epsilon [6] tools. This tool support allows the specification of CASSIS models and diagrams, the generation and validation of constraints of concept type *structures* as well as the generation of Android layout specifications, including the corresponding Java code from a CASSIS model, such as that in Fig. 2(b) and associated diagram (Fig. 3).

4 Conclusions

This paper introduced CASSIS, a tool-supported modeling language that supports customizable user interface designs. We believe that CASSIS tackles an important issue in user interface system research [10]; it supports system interaction specification in an adaptable manner. This allows a user interface design and engineering team to more effectively communicate about the specification.

[3] `http://www.iminds.be/en/projects/2014/03/05/atom`.

CASSIS was used to specify the interaction with a mobile city guide, including both design-time and runtime variation points. We were able to generate all layouts of this mobile city guide from the presented model and diagram. Generation of interactive prototypes from CASSIS models can be future work.

5 Acknowledgments

This work was partly supported by FWO project Transforming human interface designs via model driven engineering (G. 0296.08). Master thesis students Jochem Vanderheiden and Yves Bottelbergs contributed to the tool support.

References

1. Van den Bergh, J., Luyten, K., Coninx, K.: Cap3: context-sensitive abstract user interface specification. pp. 31–40. EICS '11, ACM, New York, NY, USA (2011)
2. Bongio, A., Brambilla, M., Butti, S., Comai, A., Ferronato, P., Fraternali, P., Kling, W., Molteni, E.: Interaction Flow Modeling Language (IFML) Version 0.2.3 (6 2012), OMG Doc. Nr.: ifml/2012-08-20
3. Degrandsart, S., Demeyer, S., Van den Bergh, J., Mens, T.: A transformation-based approach to context-aware modelling. Software & Systems Modeling 13, 191–208 (2014), http://dx.doi.org/10.1007/s10270-012-0239-y
4. Helms, J., Schaefer, R., Luyten, K., Vanderdonckt, J., Vermeulen, J., Abrams, M. (eds.): User Interface Markup Language (UIML) Version 4.0. OASIS (1 2008)
5. Kolovos, D., Paige, R., Polack, F.: Detecting and repairing inconsistencies across heterogeneous models. pp. 356–364. ICST, 2008, IEEE (2008)
6. Kolovos, D., Rose, L., Page, R.: The Epsilon Book. Web, http://www.eclipse.org/gmt/epsilon (2011)
7. Limbourg, Q., Vanderdonckt, J.: Engineering Advanced Web Applications, chap. UsiXML: A User Interface Description Language Supporting Multiple Levels of Independence. Rinton Press (dec 2004)
8. Mori, G., Paternò, F., Santoro, C.: CTTE: support for developing and analyzing task models for interactive system design. IEEE Trans. Software Eng. 28(8), 797–813 (2002)
9. Object Management Group: UML 2.2 Superstructure Specification (2 2009)
10. Olsen, Jr., D.R.: Evaluating user interface systems research. pp. 251–258. UIST '07, ACM (2007), http://doi.acm.org/10.1145/1294211.1294256
11. Paternò, F., Santoro, C., Spano, L.D.: Maria: A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments. ACM Trans. Comput.-Hum. Interact. 16(4) (2009)
12. Rose, L.M., Paige, R., Kolovos, D.S., Polack, F.A.: The epsilon generation language. In: Schieferdecker, I., Hartman, A. (eds.) Model Driven Architecture Foundations and Applications, LNCS, vol. 5095, pp. 1–16. Springer (2008), http://dx.doi.org/10.1007/978-3-540-69100-6_1
13. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modeling Framework 2.0. Addison-Wesley Professional, 2nd edn. (2009)
14. Van den Bergh, J., Coninx, K.: Cup 2.0: High-level modeling of context-sensitive interactive applications. In: MoDELS. pp. 140–154 (2006)
15. Vanderdonckt, J., Beuvens, F., Melchior, J., Tesoriero, R. (eds.): USer Interface eXtensible Markup Language (UsiXML). UCL (2 2012)