Model-Driven UI Engineering and Workflow Design Patterns
-A Literature Review

Addis Gebremichael

addis.gebremichael@student.ua.ac.be

Abstract

Front-end design includes expressing user interaction control-flow behaviour and data-flow of content in user interfaces of software applications. The emergence of an unprecedented mix of devices, technological platforms, and communication channels is not accompanied by novel approaches for creating a platform independent model (PIM). PIM is used to express interaction design decisions independently of implementation platform. This literature review, thereby, introduces the Interaction Flow Modeling Language (IFML), a standard front-end modeling language adopted by the Object Management Group (OMG) and employs a model-driven UI engineering paradigm. For a broader perspective of such UI modeling, this literature review essentially conducts a comparative study of the IFML with similar modeling languages in support of workflow design patterns used in the development of process-driven applications. The comparative study is also well supported with an experimental analysis by providing an IFML model representation for a BPMN model and a UML 2.0 Activity Diagram. In addition, the IFML is also experimentally evaluated for the support of UI modelling that are timed, autonomous, reactive and with dynamic structure.

Keywords: Front-end modeling, Interactive applications, Workflow systems, design patterns, process-driven application, Model-driven Engineering.

1. Introduction

The ubiquitous use of software applications, in business information systems, consumer applications, and even human-machine interfaces for industrial control, is driving powerful interaction functionalities implemented on a variety of technologies and multiple platforms. In addition, front-end design is a complex task where many requirements, perspectives, and disciplines intersect: graphic design and aesthetics, interaction design, usability, multi-screen support, offline-online usage, integration with back-end business logic and data, and coherence with the enterprise organization models (e.g., business process compliance)[1]. Thereby, front-end development continues to be an inefficient and costly process, where continuous reworking and refinement of its implementation is imposed by

the intersection of many complex factors, such as the ease to maintain and adapt, to keep up with constantly changing system requirements. Moreover, the emergence of enormous mix of devices, technological platforms, and communication channels is not accompanied by progressive approaches for creating a *Platform Independent Model* (PIM) that can be used to express the interaction design decisions independently of the implementation platforms, thus offering several benefits [1]:

- It allows for an explicit representation of the different perspectives of the UI such as structure and content of the interface, interaction and navigation options, and connection with the business logic and presentation style.
- It raises the abstraction level of frond end specifications to avoid issues addressed at the implementation level.
- It enhances team work in a development process by allowing the allocation of interaction requirement specifications and their implementation to different roles in a team.
- It provides easy communication mechanism to non-technical stakeholders, facilitating early validation of requirements.
- Given the proper tool support, it allows for model checking, not only for the correct usage of the modeling constructs but also for desirable properties of the portrayed interface such as uniformity of the interaction style and usability of the interface.

In this context, the role of a PIM - level interaction modeling language is to provide a stable set of concepts that can be used to characterize the essential aspect of the user's interaction with a software application interface: the provision of stimuli, the capturing and processing of such stimuli by the application logic, and the update of the interface based on such processing [1]. However, the PIM should be designed for change in such a way that the stable set of concepts -capturing the essential interaction aspects- should be accompanied by an extension mechanism suitable for new forms of interactions and interface renditions such as, for example, gestural stimuli and rendition on 3D devices respectively.
Additionally, common features incorporated in interactive UI modeling include the support for special features such as:

- *Timed Events*: something must happen after x time unit;
- *Autonomous*: it can trigger events and handle them autonomously;
- *Reactive*: it can react to external events, such as a user;
- *Multiple Object Instantiation*: the same object, with some designed behavior, can be instantiated multiple times with each preserving its own state;

- *Dynamic Structure*: objects can be created by the model at run-time, e.g. a new icon can appear by clicking on the canvas.
- *Model Execution*: models should not just be conceptual, analyzable models. Thus, it should be possible that UI components capturing an action or task can execute actual code, possibly by code generation.

In light of addressing the aforementioned issues, the OMG adopted the IFML as a standard for front-end modeling. IFML supports the specification of the front end of applications independently of the technological details of their realization, as discussed further in the subsequent sections. Moreover, workflow patterns provide design patterns specifically addressing recurrent problems and proven solutions related to the development of process-oriented applications [2]. These collection of patterns, such as control flow and data flow patterns, are clearly an integral part of a standard UI modeling formalism including the IFML. Hence, IFML like modeling languages such as Business Process Modeling Notations (BPMN), Activity Diagrams (AD), and ConcurTaskTrees (CTT), may provide a relative importance for such an interactive systems modeling approach. This is because an important characteristic of many process-flow/workflow models is their interactive aspect involving users' interaction flow with the underlying system in support of the process-driven application, and tasks/activities execution flow including their respective data flow. This paper, thereby, compares and evaluates the IFML with the previously mentioned three formalisms in consideration of the design patterns provided by workflow systems.
The next section gives a brief overview of the four formalisms followed by a discussion on a comparative criteria, indicating a set of evalutation criteria by which an assessment of the formalisms can be made. The subsequent section provides a thorough analysis which formulates the strength and weaknesses characterized by the used standard modeling languages. Finally, an experimental task and its findings, in light of further assessment of the comparative study, is presented followed by a conclusion to the literature review addressing the appropriateness of IFML to the task of UI modeling- in the context of a model-driven engineering paradigm.

2. Modeling Languages

In this section, for brevity purpose, a short overview about the used standards of modeling languages is provided. The basic structure, design principles, and commonly used applications or specializations of the formalisms are being introduced.

2.1. Business Process Modeling Notations- BPMN

The Business Process Modeling Notation (BPMN) is a standard notation, developed by Object Management Group (OMG)[3], for capturing business processes, especially at the level of domain analysis and high-level systems design [4]. The intent of BPMN is to standardize the business process design notation

and covers a variety of different modeling techniques which allows the creation of end-to-end business processes.
The control-flow perspective (modeling the ordering of activities) is often the backbone of a process model[5]. However, other perspectives such as the resource perspective (modeling roles, organizational units, authorizations, etc.), the data perspective (modeling decisions, data creation, forms, etc.), the time perspective (modeling durations, deadlines, etc.), and the function perspective (describing activities and related applications) are also essential for comprehensive process models[5]. BPMN constructs capture these different process modeling perspectives. In addition, BPMN provides businesses with the capability of understanding their internal business procedures in a graphical notation and will give organizations the ability to communicate these procedures in a standard manner.
Another factor that drove the development of BPMN was the need to create a bridge from the business-oriented process modeling notation to IT-oriented execution languages. Hence, this was achieved by mapping BPMN process models on to the Business Process Execution Language (BPEL), i.e. a standard for defining business processes at the implementation level [4]. BPMN process models are composed of: (i) activity nodes, denoting business events or items of work performed by humans or by software applications; and (ii) control nodes capturing the flow of control between activities. Activity nodes and control nodes can be connected by means of a flow relation in almost arbitrary ways [6]. Figure 1 shows the basic elements of BPMN.

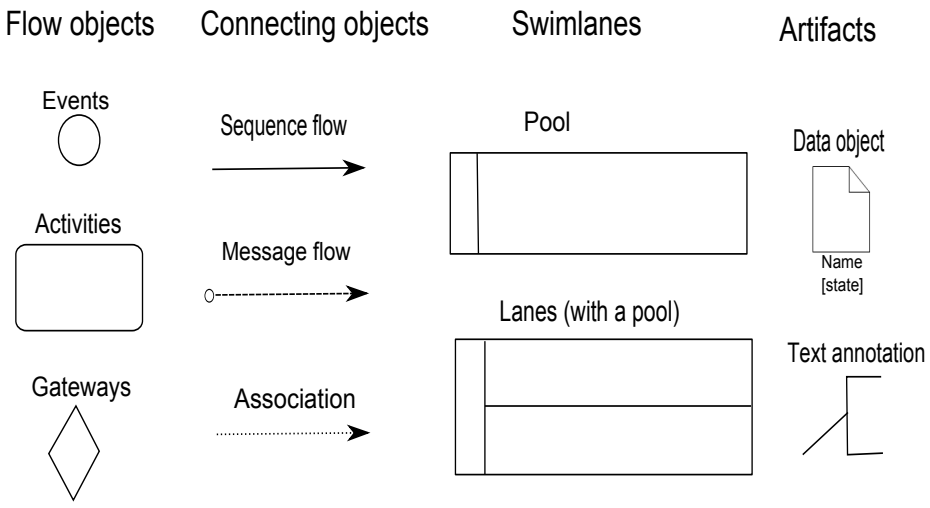

Figure 1: Overview of BPMN elements

Business process modeling is used to communicate a wide variety of information to different audiences, however, mostly suitable to design business processes at the level of business information. There are two basic types of models that can

be created with a BPMN:

- *Collaborative (Public) B2B Processes*: depicts the interactions between two or more business entities. The diagrams for these types of processes are generally from a global point of view, i.e. they do not take the view of any particular participant, but show the interactions which depict a sequence of activities and the message exchange patterns (message flows with dashed lines) between the participants indicated via *pool* constructs, as shown in figure 2. Hence, the actual (internal) processes are likely to have more activities and detail than what is shown in the collaborative B2B processes.

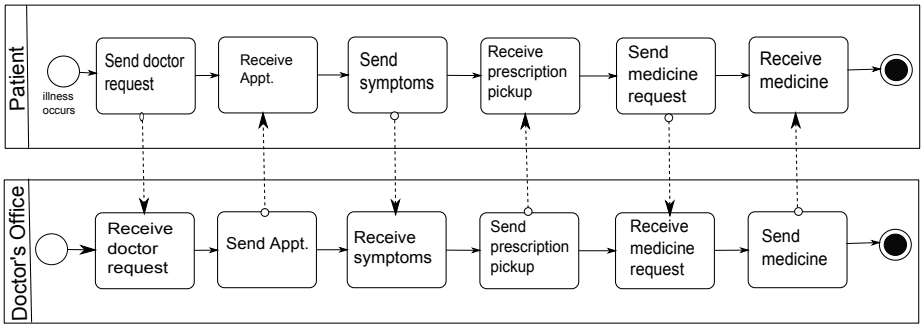

Figure 2: An Example of a Collaborative B2B Process

- *An Internal (Private) Business Processes*: generally focus on the point of view of a single business organization. Although internal processes often show interactions with external participants, they define the activities that are not generally visible to the public and are, therefore, private activities[6].

Moreover, BPMN is designed to cover many types of modeling and allows the creation of process segments as well as end-to-end business processes, at different levels of precision.The modeling of business processes often starts with capturing high-level activities and then drilling down to lower levels of detail within separate diagrams. There may be multiple levels of diagrams, depending on the methodology used for model development. Figure 3 shows an example of a simple business process model at a lower-level precision which encapsulates the "process credit card" activity as a sub-process. However, this sub-process can then be modeled differently showing the details (internal processes) at a higher-level of precision.

2.2. Activity Diagrams- AD

The activity diagram is one of the compositions of OMG's Unified Modeling Languages (UML). Activity diagram is basically a flow chart to represent the

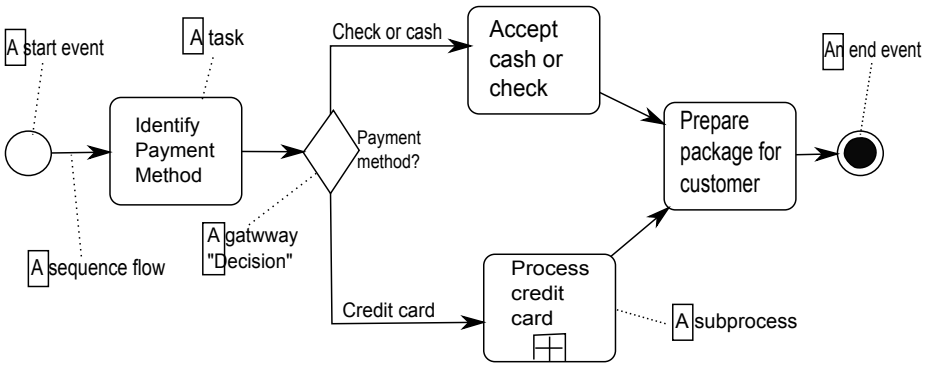

Figure 3: A simple BPMN model with basic constructs.

flow from one activity to another activity describing the dynamic aspect of a system. Activity is a particular operation of a system. Thus, the purpose of the activity diagram is to model the procedural flow of actions that are part of a larger activity.
In UML 2.0 AD the fundamental unit of behaviour specification is *Action*. An action takes a set of inputs and converts them to a set of outputs, though either or both sets may be empty[7]. Actions may also modify the state of the system. There are three types of actions as follows:

- *Invocation Actions*: used for performing operation calls, signal sending and accept event actions;
- *Read and Write Actions*: used for accessing and modifying objects and their values;
- *Computation Actions*: for transforming input values into output values.

Although the language provides further a very detailed action types, here we focus on the basic ones as it will be out of the scope of this paper. These are, as shown in figure 4a, the generic *Action* concept, *Accept Event Action*, *Send Signal Action*, and *Call Behavior Action*. Furthermore, the concept of *Activity* is used to present the overall behaviour of a system. Activities are composed of actions and/or other activities and they define dependencies between their elements. With respect to their visual concrete syntax, they are composed of nodes and edges. The edges, used for connecting the nodes, define the sequential order among them. Nodes represent either *Actions*, *Activities*, *Data Objects*, or *Control Nodes*. The various types of control nodes are shown in figure 4b.
Besides modeling the procedural flow of control for various activities, one main reason to include activity diagrams in an overall system model is that they model the data carried by most control flows, which is similar to BPMNs but with better semantics and expressive power. In ADs it is possible to describe the data passing in and out of an activity in either of two ways [8]. The first one

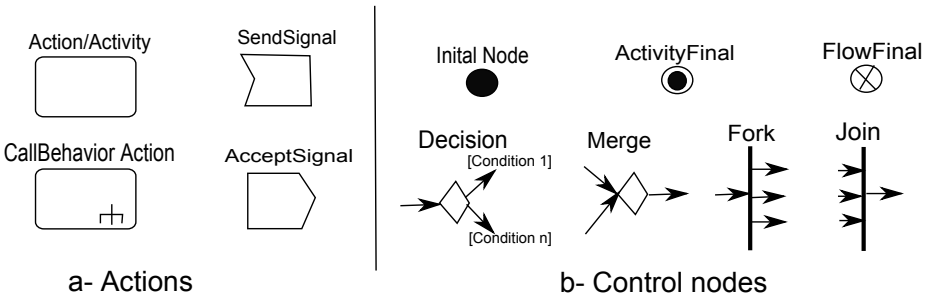

Figure 4: Activity diagram modeling elements

is by using **Object Nodes**, as the simplest method of describing the information flowing between activities. An object node is like a variable in a program. It represents something that stores one or more values that are passing from one action to another. Another way to describe data flow in ADs is by using an **Output Pin** and an **Input Pin** which enables to separately describe the outputs from one action and the inputs to another. Pins are like parameters in a program. Pins represent ports where objects can enter and leave an action. Hence, the significance of this sort of model is that it allows modelers to get a better understanding of the data aspect that could possibly influence the control flow of activities. Figure 5 shows the concrete syntax for data flows in ADs: UML 1.5 notation (left), and in alternative equivalent UML 2.0 notations (all others), including attached data flow, and pin notations (third and fourth).

Because it models procedural flow, the activity diagram, which is also a vari-

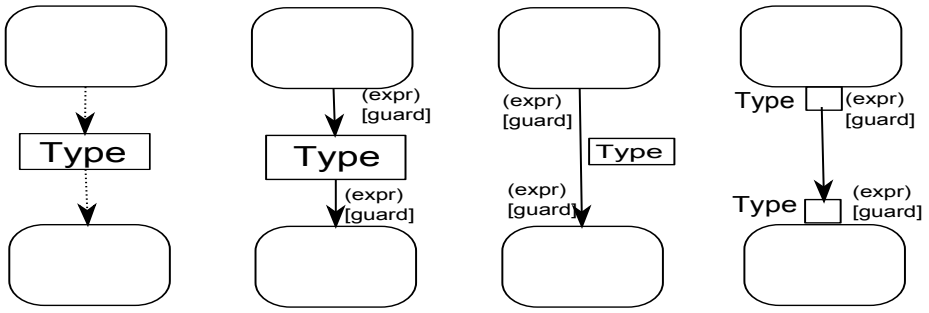

Figure 5: Data flow in Activity Diagrams

ation of statechart diagrams, focuses on the action sequence of execution and the conditions that trigger or guard those actions. It is possible to use Post-conditions and Pre-conditions properties to specify in detail the outcome of an action. These properties describe the effect of the action without describing how the effect is achieved. Moreover, similar to BPMNs which enable a high-level precision model as discussed in the previous section, ADs also describe the de-

tailed behavior of sub-activities with "Call Behavior Actions", using a separate activity diagram. A called behavior is an activity diagram that is represented on the main activity diagram by a Call Behavior Action as illustrated in the figure 6. Moreover, unlike BPMNs, the activity diagram is also focused only on

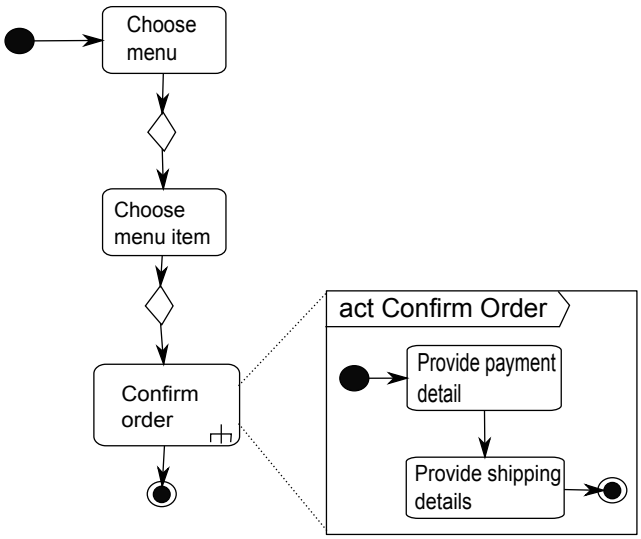

Figure 6: Describing Sub-Activities with Call Behavior Actions in ADs

the activity's internal actions, which is more suitable for the design of complex system behaviors, and not on the actions that call the activity in their process flow or that trigger the activity according to some event, for example.

Thus, activity diagrams are more suitable to:

- describe a business process or a flow of work between users and a system.
- describe the steps performed in a particular use case. In projects in which use cases are present, activity diagrams can model a specific use case at a more detailed level. However, activity diagrams can be used independently of use cases for modeling a business-level and/or system-level behaviors [9].
- describe a method, function or operation in a software.

2.3. ConcurTaskTrees -CTT

Task models represent the intersection between user interface design and more engineering approaches by providing designers with a means of representing and manipulating a formal abstraction of activities that should be performed to reach user goals [10]. For this purpose they need to be represented with notations, such as *ConcurTaskTrees*, able to capture all the relevant aspects. ConcurTaskTrees provides a notation for task model specifications useful to support

design of interactive applications specifically tailored for user interface model-based design.
By focusing on users and their tasks, task models provide design principles used to obtain usable interactive system and play an essential role representing the logical activities that an application should support in reaching users goals [10]. A goal is either a desired modification of state or an inquiry to obtain information on the current state. Each task can be associated with a goal which is the state change derived by its performance. Each goal can be associated with one or multiple tasks since there can be different tasks to achieve the same goal. Tasks can range from a very high abstraction level, such as deciding a strategy for solving a problem, to a concrete- action-oriented level, such as selecting a printer device. Basic tasks are elementary tasks that cannot be longer decomposed because they do not contain any control element.
To illustrate some of the basic concepts discussed above, an example of making a hotel reservation is a task that requires a state modification, i.e. creating a new reservation in the hotel's persistence repositary- such as a database. However, as in figure 7, this task can be decomposed into lower level tasks. It is important to remember that hierarchy (task decomposition) does NOT represent sequence: the sequential temporal evolution is represented linearly from left to right instead of from top to down. Hence, the task model in figure 7 has an execution order which can be read as follows: In order to make a hotel reservation, the task of selecting room type must be done followed by making the actual reservation, i.e. when the system shows available rooms and the task of selecting a particular room is attained. Also to note is that querying the available rooms is one task that just requires an inquiry of the current state of the application. There can be many motivations for developing a task model. In some cases

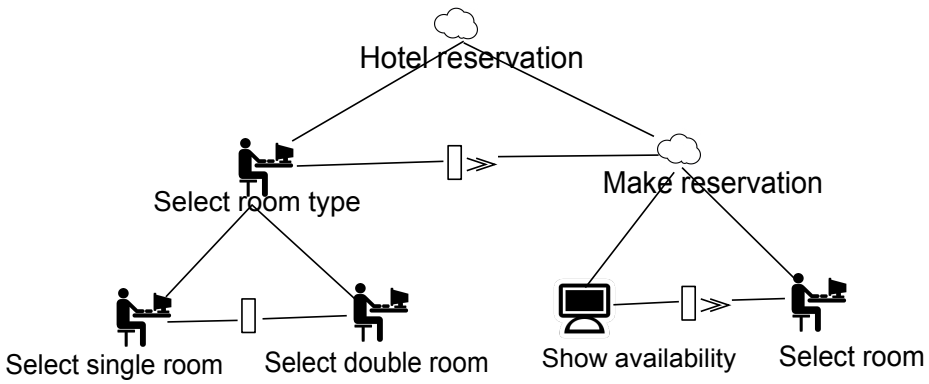

Figure 7: An example of a simple task model with hierarchical structure

where a task model is developed for an existing systems, its purpose is to better understand the underlying design and analyse its potential limitations and how to overcome them. While, in other cases, designers create task models for new applications yet to be developed. Here the purpose is to indicate how activities

should be performed in order to obtain a new, usable system that is supported by some new technology.
The main features of ConcurTaskTrees are as follows [10]:

- *Focus on activity*: It allows modelers to concentrate on the activities that users aim to perform, that are the most relevant aspects when designing interactive applications that encompass both user and system-related aspects avoiding low-level implementation details that at the design stage would only obscure the decisions to take.

- *Hierarchical structure*: It provides a wide range of granularity allowing large and small task structures to be reused, it enables reusable task structures to be defined at both low and high semantic level. Moreover, it is an intuitive way of problem solving by way of decomposing and still maintaining various parts of the solution domain.

- *Rich set of temporal operators*: The graphical syntax supports for operators which capture a rich set of possible temporal relationships between the tasks that can be defined. Figure 8 shows an example of two operators and a description of their semantics.

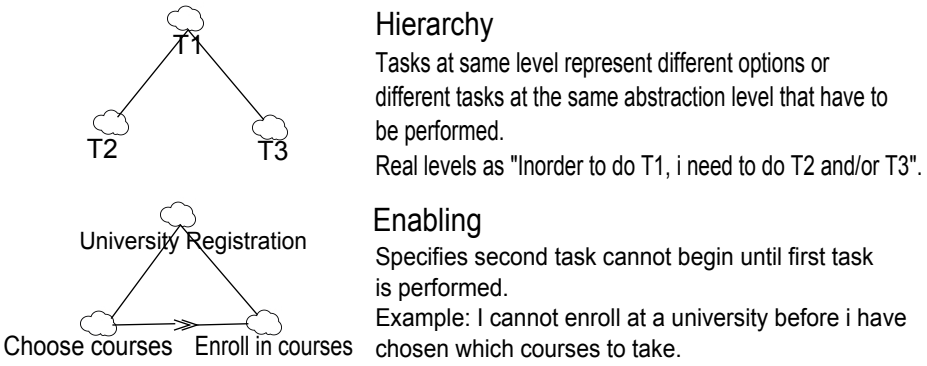

Figure 8: An example of temporal operators provided by CTT

- *Task allocation*: How the performance of the task is allocated is indicated by the related category and it is explicitly represented by using icons.

- *Objects and task attributes*: Once the tasks are identified it is important to indicate the objects that have to be manipulated to support their performance. Two broad types of objects can be considered: the user interface objects and the application domain objects. For example, Temperature as a domain object can be represented by multiple UI objects such as a bar-chart or a textual value. Another important attribute is the *Platform Attribute*, which allows designers to specify for what type of platform the task is suitable. In addition, it is also possible to specify for each object

manipulated by the task in which platform it is suitable to support them. This type of information allows designers to better address the design of nomadic applications that can be accessed through a wide variety of platforms.

Furthermore, similar to BPMN and ADs, task models also support for cooperative applications, i.e. the design of applications where multiple users can interactively cooperate. In this approach as in figure 9, when there are multi-user applications, the task model is composed of various parts. A role is identified by a specific set of tasks and relationships among them. Thus, there is one task model for each role involved. In addition, there is a cooperative part whose purpose is to indicate the relationships among tasks performed by different users.

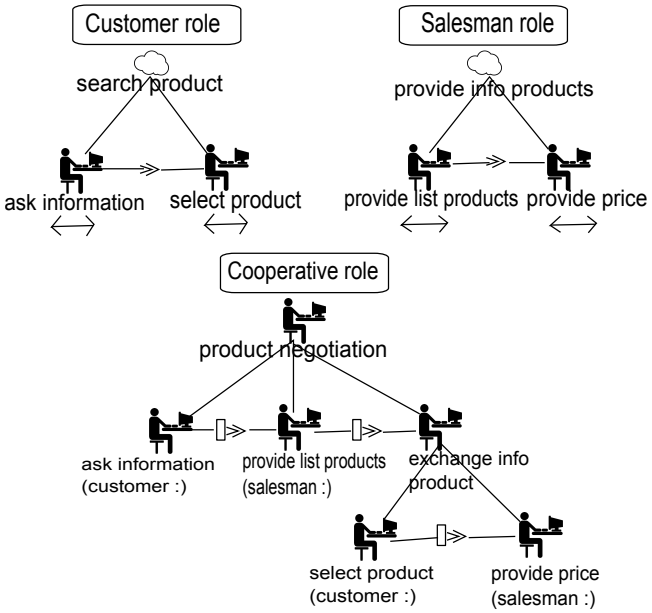

Figure 9: Simplified example of cooperative task model.

Thus, task models provide better incentive for the design and development of interactive systems, i.e. human-computer or multiple cooperative applications, since they look at user interface development from the logical activities that have to be carried out in order to reach the users goals. Moreover, if a direct correspondence is created between the tasks and the user interface, then users can accomplish their goals effectively and satisfactorily.

2.4. Interaction Flow Modeling Language -IFML

The Interaction Flow Modeling Language (IFML), also an OMG standard, supports the specification of the front-end of applications independently of the

technological details of their realization. In general, it attempts to address the following aspects of front-end modeling [1]:

- *The composition of the view*: It expresses what the visualization units that compose the interface are, via ViewContainer elements, and their nesting relationships, visibility and reachability.
- *The content of the view*: It specifies what the elements for content display and/or accepting user input are, via ViewComponents elements. Examples of ViewComponents construct are interface elements for visualizing the data of one object, for displaying a list of objects, data entry forms for accepting user input. These elements are further characterized by a ContentBinding specification which expresses the source of the published content; i.e. it can originate from objects referenced in a domain model such as Entity-Relationship diagram or from an external service such as a web API service invocation to receive content for publishing.
- *The commands*: It realizes what interaction events are supported. Events affect the state of the user interface and can be produced by the user's interaction, by the application itself, or by an external system.
- *The actions*: It specifies what business components are triggered by the events.
- *The effects of interaction*: It captures what the effect of events and action execution bring on the state of the interface.
- *The parameter binding*: It expresses what data items are communicated between the elements of the user interface and the triggered actions/events.

The above discussed aspects of front-end modeling are captured by IFML modeling constructs as shown in figure 10. Designing a modeling language for the front end is a complex and multidisciplinary task where many perspectives meet. Thus, a good modeling language should pay attention to coverage, i.e. the ability to represent complex application front ends but also to model usability and understandability which contribute to make the modeling language quick to learn, simple to use, easy to implement by a given tool, and open to extensibility [1]. The design of IFML adheres as much as possible to the following principles [1]:

- *Conciseness*: The number of diagram types and concepts needed to express the salient interface and interaction design decisions is kept to the minimum. In particular, the IFML visual syntax conveys the front-end model using a single diagram with the help of compositional nesting design structure. This design simplifies the model editing and maintenance processes, because references between different types of diagrams need not be maintained and only the internal coherence among the various elements of a single type of diagram must be preserved.

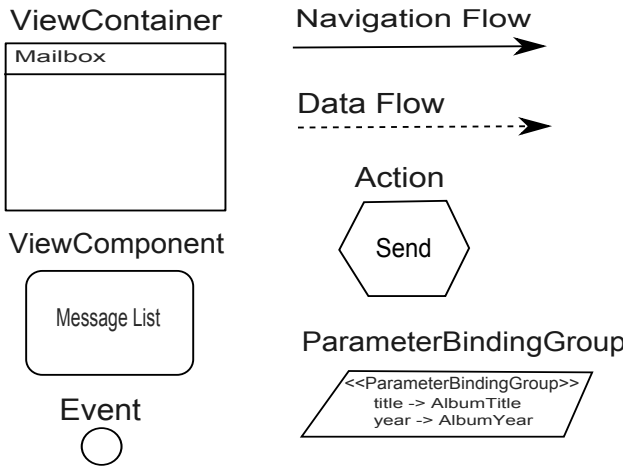

Figure 10: IFML basic modeling elements.

- *Inference from the context*: Whenever something can be deduced from the existing parts of a model, inference rules at the modeling level automatically apply default modeling patterns and details, avoiding the need for modelers to specify redundant information. An example can be specifying parameter binding rules between two different ViewComponent elements, referring two different objects in a domain model, for content display. Thus, during data binding among the ViewComponents via an attached ParemeterBinding element, the modeler need not specify properties of an object for data transfer since the ParameterBinding element can easily infer object attributes from the attached ViewComponents.

- *Extensibility*: IFML builds upon a small set of core concepts at a higher-level of abstraction that capture the essence of interaction: the interface (container), stimuli (events), content (components and data binding), and dynamics (flows and actions). However, by design, these concepts are meant to be extended to mirror the evolution of technologies and devices such as the specifications of web, desktop, and mobile applications, which the IFML supports.

- *Implementability*: Models that lack adequate tool support and cannot be executable are quickly abandoned. Hence, IFML is a platform-independent language but has been designed with executability in mind through model transformations and code generators to ensure models can be mapped easily into executable applications for various platforms and devices.

- *Not everything in the model*: IFML purposely ignores presentation aspects, because presentation is adversarial to abstraction. It also delegates

to external models the specification of aspects that although relevant to the user interface, are not properly part of it. For example, the internal functioning of an action can be described by referencing a method in a UML class diagram.

Hence, IFML is a platform independent user interface modeling language for the realization of a user's interaction flow in an application's front-end. The main focus is on the structure and behavior of a software application as perceived by the end user. In addition, IFML also incorporates references to the data and business logic that influence the user's experience. This is achieved respectively by referencing the domain model objects that provide the content for display, and the actions that can be triggered by the user interface which refer to methods of the application domain objects, described by suitable UML behavioral diagrams, or delegated to external objects and services.

3. Comparative Criteria

As discussed previously, the four formalisms reflect design patterns inherent to process-driven applications, hence, relationships among them can be drawn based on categories of evalution criteria. In the next section, an analysis of the four modeling languages with respect to their support for elaborated key aspects is discussed. Thus, the focus will be to draw their differences and commonalities concerning the selected evaluation criteria.
There can be collection of key functionalities that are supported by the commonly used formalisms. This paper has identified five key aspects which we will concentrate on to broaden our focus. To provide a comprehensive support for modeling process-oriented applications with workflow patterns, the four formalisms previously introduced are compared for the realization of these patterns and other key aspects. In this section, an introduction of the identified five key aspects for evaluating the formalisms is provided.

3.1. Functional

The functional aspect concentrates on what has to be done in a workflow process. It defines hierarchical context of tasks and sub-tasks. To analyze the support for this functional aspect by the four notations, we compare the possibility for nesting constructs.

Possible values are: **Yes**, **No**.

Another functional aspect is related to constraints of workflow patterns. These are sets of rules covering consistency. These are similar to constraints on workflow-nets and are of three kinds [11]: Enter constraints are being evaluated before the initiation of a workflow process. Run-time constraints are verified during execution of the process and is discussed in detail in the behavioral aspect section. Exit constraints are checked at the completion time of a workflow.

Possible values are: **Enter**, **Exit**, **Both**.

3.2. Behavioral

The behavioral aspect concentrates on the control flow perspective of the workflow model. Execution order of activities in a workflow and/or interactive components of a system is of great interest. Thus, an in depth analysis of all provided patterns for all four notations would go beyond the scope of this paper. Thereby, support for the basic control flow patterns such as the ones illustrated next are assessed for the previously discussed four formalisms.

- *Sequential Execution*: A task in a process in enabled after the completion of a preceding task in the same process which denote an ordering relation.

  Possible values are: **Yes**, **No**.

- *Exclusive Choice (Decision)*: The divergence of a branch into two or more branches such that when the incoming branch is enabled, the thread of control is immediately passed to precisely one of the outgoing branches based on a mechanism that can select one of the outgoing branches.

  Possible values are: **Yes**, **No**.

- *Synchronization*: The convergence of two or more branches, which are executing in parallel or in any order, into a single subsequent branch such that the thread of control is passed to the subsequent branch when all input branches have been enabled. The input branches can perform in parallel (at the same time) or in any order.

  Possible values are: **Parallel Synch.**, **Non-parallel Synch**, **Both**, **None**.

- *Simple Merge (XOR join)*: The convergence of two or more branches into a single subsequent branch such that each enablement of an incoming branch results in the thread of control being passed to the subsequent branch. Hence, the simple merge pattern provides a means of merging two or more distinct branches without synchronizing them.

  Possible values are: **Yes**, **No**.

- *Parallel Split (AND-split)*: The divergence of a branch into two or more parallel branches each of which execute concurrently.

  Possible values are: **Yes**, **No**.

- *Exception Handling*: Deals with the various causes and the resultant actions needed to be taken as a result of exception occurrences.

  Possible values are: **Yes**, **No**.

- *Multiple Instantiation*: Multiple instance patterns describe situations where there are multiple threads of execution active in a process model which relate to the same activity, i.e. the activity is executed multiple times with each of these instances running concurrently and independently of

the others. The number of instances (n) may be determined at design time or at run-time.

Possible values are: **Design-time**, **Run-time**, **Both**, **None**.

- *Iteration*: It specify special behaviour such as repetition, i.e. the activity/task is executed multiple times sequentially.

  Possible values are: **Yes**, **No**.

- *Message exchange modality*: The emergence of service-oriented computing has led to a significant shift from traditionally tightly coupled to loosely coupled software development paradigm. Software components, such as a given business logic or the user interaction component, are exposed as a service where each service provides its functionality via a platform independent interface. Thus, message exchange is the only way to communicate with a certain service. Nonetheless, this communication is made possible in two different forms which influence the control-flow behavior of a system, namely synchronous and asynchronous.
  In a process driven service oriented architectures (SOA), a synchronous service invocation follows the well-known Request / Response Pattern where a process engine performs a synchronous request to a service and waits for a response, i.e. it's blocked until the response is obtained. While an asynchronous service invocation usually follows the Request / Acknowledge / Callback Pattern where the process engine sends the request to the service but instead of processing the request right away, the service first puts the request into a Queue. It then acknowledges the receipt of the request to the process engine. The request is then delivered asynchronously to the actual request processor. The request processor executes the business logic. Finally it sends a callback to the process engine which may now continue execution in the process instance.
  Hence, the purpose of this evaluation criteria is to assess the formalisms' support for such message exchange modalities affecting the control-flow perspective of a process model.

  Possible values are: **Synchronous**, **Asynchronous**, **Both**.

- *Interruptibility*: In a process flow, the sequential execution of activities can be interrupted due to internal/external events where internal event include a timer and/or system event while external ones can be user's interaction with a system. This can of course influence the control flow of a process model since "normal" execution order can be reoriented. Hence, the ability to capture such a behavior by the four formalisms is illustrated in this assessment.

  Possible values are: **Yes**, **No**.

3.3. Informational

The integral part of this aspect is data which is being processed in a workflow process. From a data perspective, there are a series of characteristics

that occur repeatedly in different workflow modeling paradigms. These can be divided into four distinct groups as follows: [12]

- *Data visibility*: relates to the extent and manner in which data elements can be viewed by various components of a workflow process.

  Possible values are: **Yes**, **No**.

- *Data interaction*: focuses on the manner in which data is communicated between active elements within a workflow process.

- *Data transfer*: considers the means by which the actual transfer of data elements occurs between workflow components and describes the various mechanisms by which data elements can be passed across the interface of a workflow component.

  Possible values for data interaction and transfer are: **Low** (if there is no modeling constructs denoting data transfer/interaction but it's assumed implicitly), **Average**, **High** (if there exists rich set of modeling constructs for both data interaction and transfer).

- *Data based routing*: characterises the manner in which data elements can influence the operation of other aspects of the workflow, particularly the control flow perspective.

  Possible values are: **Yes**, **No**.

The interested reader may refer to [12] for detailed workflow data patterns. The purpose of this analysis is to assess if the previously discussed formalisms support for any of the above mentioned data flow characteristics.

3.4. Level of Abstraction

An abstraction level is a generalization of a model or algorithm, away from any specific implementation to facilitate interoperability and platform independence. The availability of an increasing number of device types ranging from web applications to mobile and desktop is changing the structure of many applications. It is often the case that within the same application users can access its functionality through different devices. Hence, context of use is acquiring an increasing importance and it is important to understand its relationships with modeling languages.

Often standard formalisms possess an extension of their core modeling concepts to support for multiple platform applications, thereby adding more expressive power to the modeling language and making the concepts and notations less abstract. Moreover, it assigns more precise meaning to concepts to enable deeper model checking, formalization of semantics, and executability through code generation or model interpretation. Thus, the purpose of this analysis is to determine if one formalism is more or less platform specific or provides a low-level abstraction than the other, or can be suitable to model at various abstraction levels also enabling code generation.

Possible values are: **Low-level**, **High-level**, **Both**.

3.5. Tools Support

Modeling languages that lack adequate tool support and cannot be used to produce executable code are quickly abandoned. Both at the commercial and research levels, several tools for MDE are either available or in development. These tools are commonly designed as frameworks or as plug-in. Hence, the purpose of this evaluation criteria is to assess, at the time of publication, if a given formalism is implemented with a tool that facilitates model constructing, model checking, and code generation.

Possible values for implementation are: **None**, **Partial** (formalism has a prototype tool but lacks essential features which can be underdevelopment), **Complete**.

Moreover, it is also imperative to assess if the available tools have adequate documentation and community involvement for maintenance purposes.

Possible values for maintenance are: **None**, **Low**, **High**.

3.6. Integration with other formalisms

Model based approaches are becoming more predominant in software engineering. Looking at the most successful model-based approach for design of software systems, we can notice a considerable effort to provide models and representations to support the various phases and parts of the design and development of software applications. However, many formalisms are specialized to represent certain domains, hence do not allow for a comprehensive system modeling and/or do not easily support code generation due to their high-level of abstraction. Thus, to exploit a particular modeling language's expressive power for a comprehensive system design and executability, it must be put in context within a broader modeling perspective. Thereby, the purpose of this analysis is to assess the possibility of integrating and its actual implementation of the previously discussed formalisms among each other and/or with a standard software engineering modeling paradigm, such as the UML, to support for an end to end system design or possibly to facilitate code generation by mapping the formalism onto low-level executable languages.

Possible values are: **Name of formalism/Implementation Tool**

4. Analysis

In this section, the given set of evaluation criteria discussed in the previous section is used to assess the four formalisms. An overview of the obtained results is illustrated in Table 1.

Criterion / Formalism	BPMN [3] [4] [5] [6] [2]	AD [7] [8] [9] [13]	CTT [10] [14] [15]	IFML [1]
Functional Aspect				
Compositional nesting	Yes	Yes	Yes	Yes
Workflow pattern constraints	Both	Both	Both	Both
Behavioral Aspect				
Sequential execution	Yes	Yes	Yes	Yes
Exclusive choice	Yes	Yes	Yes	Yes
Synchronization	Parallel Synch.	Parallel Synch.	Both	Both
Simple merge	Yes	Yes	Yes	Yes
Parallel split	Yes	Yes	Yes	Yes
Exception handling	Yes	Yes	No	Yes
Multiple instantiation	Both	Both	None	Design-time
Iteration	Yes	Yes	Yes	Yes
Message exchange modality	Both	Both	Synchronous	Synchronous
Interruptibility	Yes	Yes	Yes	Yes
Informational Aspect				
Data visibility	Yes	Yes	Yes	Yes
Data transfer and interaction	Average	High	Low	High
Data-based routing	Yes	Yes	Yes	Yes
Level of abstraction	Both	Both	Both	Both
Tools support				
Implementation	Complete	Complete	Complete	Partial
Maintenance	High	High	Low	Low
Integration with other formalism				
Comprehensive design support	IFML/WebRatio	CTT/Eclipse task editor	CD/CTTE	BPMN/WebRatio
Code generation	BPEL/BPMN2BPEL	CD/Enterprise Architect	TERESA/CTTE	-/WebRatio

Table 1: An overview of the comparative study based on the set of evaluation criteria.

4.1. Functional Aspect

Looking into BPMN and AD, both support for nested hierarchy through sub-process and sub-activity modeling constructs respectively. They can be viewed as a standalone (atomic) sub workflows. Hierarchical structure also comes very intuitively to task models constructed using concurTaskTree notations. A task model usually has a tree like hierarchical structure which reflect the nesting of a set of tasks and traversing of a logical inverted tree. IFML, on one hand, also supports for nested hierarchy through a compositional nesting of the ViewContainer notation. The set of existing ViewContainer elements can be represented as an AND-OR tree representation which facilitates the analysis of a ViewContainer's state as visible and reachable.
The other functional aspect is related to constraints of a workflow process. Both BPMN and AD support for a pattern in workflow nets where the start of a workflow is triggered via an entry constraint such as a start/message event and terminates with an exit constraint represented via an end event. Task models have a logical tree structure where the sequential temporal evolution is represented linearly from left to right. Thus, the most left atomic task can trigger the entire task model and can possibly end with the right most atomic task. In IFML, a tree representation of the ViewContainer construct can represent the root ViewContainer which is accessed first (visible initially) and the internal children nodes with their own access modifiers as entry/exit constraints. Runtime constraints verify what happens during process execution, hence, all four formalisms have a rich set of notations in support of that which will be discussed in the following two sections.

4.2. Behavioral Aspect

As discussed above, the behavioral aspect focuses on the control flow perspectives of a workflow process. Hence, we will assess the support of the previously discussed four formalisms for the basic control flow patterns starting with sequential execution.

- *Sequential Execution*: All four formalism have a rich set of modeling constructs in support of sequential flow which links two objects in a process diagram and denotes a control flow (i.e. ordering) relation.

- *Exclusive Choice (Decision)*: This kind of control flow patterns are handled via "Decision-gateway" constructs in BPMN. Activity Diagrams also have "Decision-node" constructs in support of this. In ConcurTaskTrees notation, "Choice Operator" has a similar semantic behavior in support of conditional branching as indicted in figure 11. Moreover, IFML captures what the effect of events and action execution bring on the state of the interface. Hence, IFML supports for two types of interaction flow patterns, namely content based and content independant navigation. In both cases, the execution of an event or action can trigger a navigation flow in support of conditional branching (decision) pattern. An example of a content dependent navigation is shown in figure 12, where the events

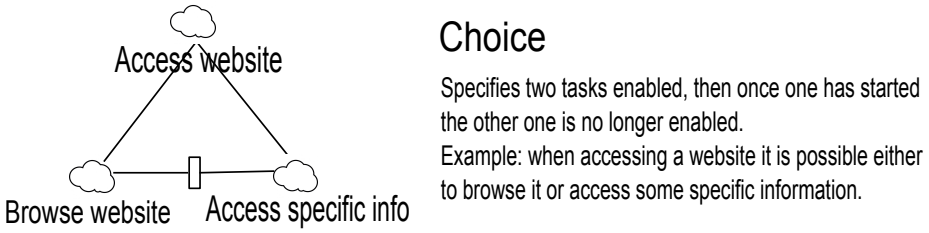

Figure 11: Choice Operator in ConcurTaskTree notation

attached to "ArtistsAlbum" ViewComponent can change the state of the interface either to an "Artists or Album" ViewComponent contained in the corresponding ViewContainer.

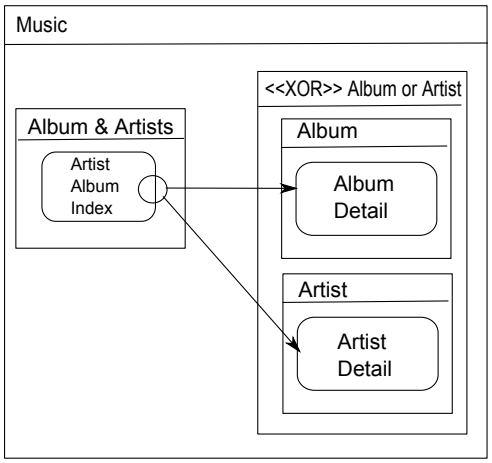

Figure 12: Decision pattern in IFML

- *Synchronization*: Both BPMN and Activity Diagrams have constructs such as "Parallel-join gateway" and "Join node" respectively, to represent when parallel action sequences terminate the elements are used to indicate that the multiple threads are joined back into a single thread. In Petri-net terms, its semantic behavior can be expressed as in the figure 13, where the transition is enabled if tokens are present in both places. Similarly, ConcurTaskTrees notations have a Concurrent Tasks operator which indicates that sub-tasks can be performed in any order, or at same time (in parallel), but the task model is completed as long as all sub tasks are performed.
  In IFML, the effect of interaction can also be recognized following a synchronization pattern, given input branches can perform concurrently or in

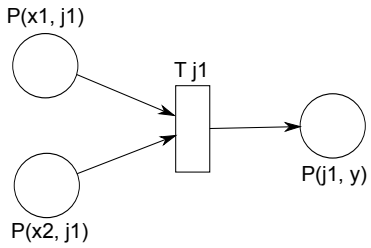

Figure 13: Perti Nets Formal Semantics for Synchronization

any order (non-parallel). For example, in figure 14, the Action "Execute the payment" is triggered only when an input of total amount is passed via the specified data flow (dashed arrow) directed to it and during the submission of "Payment information form" via its incoming navigation flow.

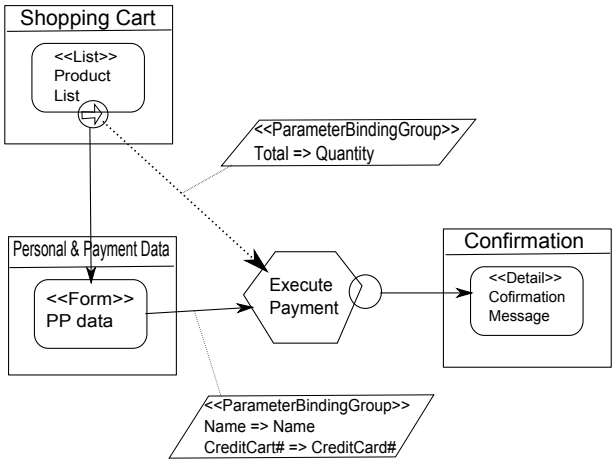

Figure 14: Synchronization pattern in IFML

- *Simple Merge (XOR join)*: "Parallel-merge gateway" and "Merge node" constructs capture this kind of semantic behavior in BPMN and Activity diagram respectively. In Petri-net terms, its semantic behavior can be expressed as in the figure 15, where either of the transitions is enabled if a token is present in either of the corresponding places. Optional tasks have a subtle semantics in ConcurTaskTrees. They can be used only with concurrent and sequential operators. Their name is closed between squared brackets to indicate the sub-tasks are optional, implying there is no synchronization.
  In IFML, a simple merge pattern can be easily recognized, for instance,

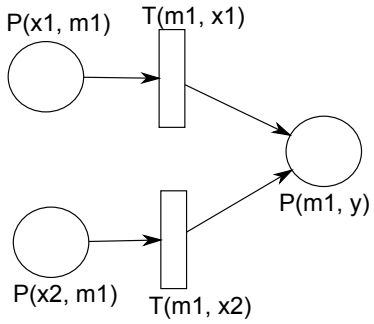

Figure 15: Perti Nets Formal Semantics for Simple Merge

in a scenario where a log-out action is triggered either when a user interaction event such as clicking on a log-out button takes place or during a system/timer event is triggered indicating an expiration of a session.

- *Parallel Split (AND-split)*: Both BPMN and Activity Diagram support for this behavior by their "Fork gateway" and "Fork node" constructs respectively. Note that these constructs are introduced to support parallelism in activities. Thus, in Petri-net terms, its semantic behavior can be expressed as in the figure 16, where during the transition is enabled and it fires, tokens are distributed to the corresponding places to imply the occurrence of concurrent activities/tasks. ConcurTaskTrees notations also

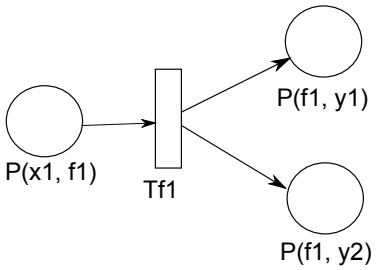

Figure 16: Perti Nets Formal Semantics for Parallel Split

provide similar semantic definition for parallel split pattern via its "Independence Concurrency" and "Concurrency with information exchange" operators. In the first one, the tasks can be carried out in any order or at the same time (concurrently) howeover, in the second one, sub-tasks are actually executed concurrently given that they have to synchronize in order to exchange information.
Moreover, IFML constructs also support for parallel split pattern in various ways. For example, in figure 14, when a user triggers the checkout

event, a navigation flow changes the state of the user interface to the customer information "ViewContainer" where a form is available to fill in. At the same time, a data flow capturing the total amount selected is passed as an input to the action "Execute the payment".

- *Exception Handling*: BPMN and ADs support for "Error events" which trigger as a result of a run-time exception occurance. Hence, they both support for "Exception-flows" which determine the sequential flow of a process model. The formal semantics of exception handling can be best represeted by "Decision" control-flow perspective, where conditional branching takes place indicating the possible condition that determines the cause of the error.
  In CTT notation, there is no modeling construct in support of such pattern. Hence, CTT lacks exception handling mechanism which is also an essential pattern influencing the control-flow perspective of a model.
  IFML, however, has "Error events" similar to BPMN and ADs. These events are part of IFML's "Action" element which determine the flow of navigation if an exception occurs while executing operational aspects of the action.

- *Multiple Instantiation*: In both BPMN and Activity diagrams, there is no major difference in diagramming this pattern. Both notations provide mechanisms for creating the variations (at design-time or run-time) of this pattern. For run-time pattern, both have a "Multiple-instance" construct, with an input/output collection attached to it, that can *dynamically* determine how many copies of the contents of the element will be performed before the process continues. For example, in a process model for diagnosing an engine fault, multiple instances of a "Check-sensor" task can run concurrently depending on the number of error messages received, determined at run-time. Only when all messages have been processed, can the "Identify-fault" task, i.e. the next activity, be initiated. The number of error messages received can serve as an input collection for the modeling construct, which can then determine at run-time how many duplicates of the task will run in parallel.
  The number of elements can also be defined *statically* at design time which then replicates the activity involved as many times as specified. Thus, this kind of multi-instance activity can be replaced by n identical copies of the activity enclosed between an AND-split and an AND-join, a 'Macro' expansion of the modeling construct, as shown in Figure 17.
  In CTT notation, there is no notion of multiple instantiation of a task. However, tasks can have a looping structure denoting an iteration as indicated in the next section.
  The IFML constructs support the semantic behavior reflected by multiple instantiation of an object determined at design-time only. For instance, a modeler can specify multiple "ViewComponent" elements of n, which refer to the same domain object, and these elements can be linked with a

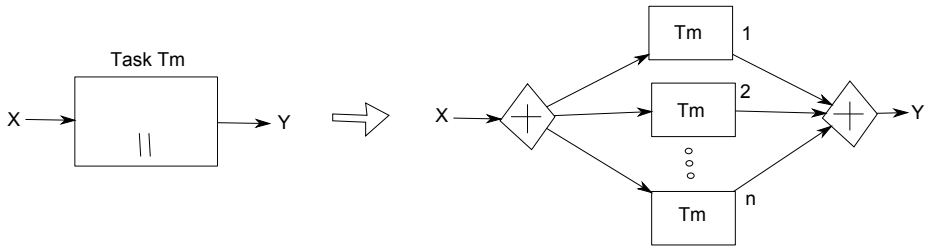

Figure 17: Macro expansion for a multi-instance activity where n is known at design time

navigational flow to multiple events of n. Hence, the triggering of each attached event can make visible/active its corresponding "ViewComponent" which instantiates the refereced domain object. Thus, multiple copies of the same object can co-exist at a certain point in time.
However, there is no notion of dynamic instantiation of the same object multiple times when only considering IFML constructs. Since the "Action" element in IFML does not enable for specifing its internal behavioral aspects, it can refer to methods of the application domain objects, described by a suitable UML class diagram for example. Thereby, the referenced method of an object can specify such a behavior where it can dynamically instantiate multiple objects of the same type. This happens as a result of triggering the "Action" element multiple times by a corresponding event which then invokes a method of the same object that receives some parameter values to instantiate it.

- *Iteration*: Both BPMN and ADs support for the repetition constructs (iteration) can be seen as "macros", in the sense that they can be expanded in terms of decision and merge nodes as shown in Figure 18. Note that the value of attribute "TestTime" determines whether the repeated activity corresponds to a "while" loop or a "repeat-until" loop. In ConcurTask-

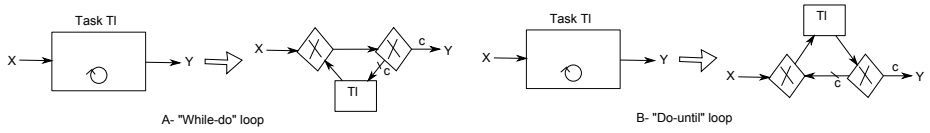

Figure 18: Macro expansions for repeated activities

Trees notation, this pattern is only represented in an iterative way, i.e. the task model is executed multiple times sequentially. For example, in Figure 19 once the task "CloseCurrentNavigation" is performed then the task "SelectMuseum" is enabled again because the parent task is iterative, denoted by an asterisk following the task name. This means that it can be performed multiple times and so its first sub-task is enabled again once

the last one has been completed. Morever, IFML also supports for this

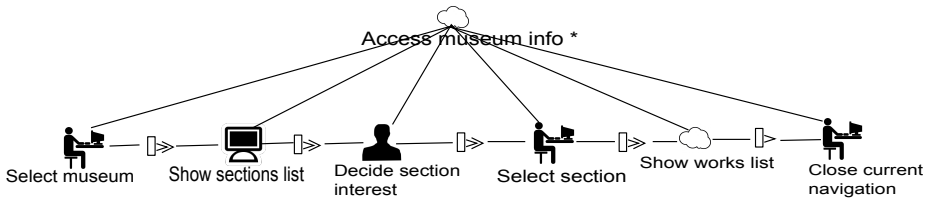

Figure 19: Example of parent iterative task in CTT

control-flow pattern in a manner where, for example, a log-in terminal at the user interface gives the chance of authenticating user's credential n number of times. Thus, a log-in event can trigger an "Action" which refers to some external service component that verifies user credentials and returns succuss or failure. If the result is a failure, an error event attached to the action element triggers to change the navigational flow to the same "Log-in ViewComponent" activating it again. This sequence of interaction loops until n number of times verified by a "ConditionalExpression" specificiation of the "Log-in ViewComponent".

- *Message exchange modals*: The BPMN 2.0 Specification provides the Service Task activity allowing users to invoke some application service in both synchronous and asynchronous manner. The UML 2.0 AD Controlflows or objectflows are connecting activities and actions in synchronous processes - determining the flow through possible subsequent working steps. By usage of signals (Send Signal Action, Receive Signal Action and Timer Action) processes can be uncoupled, can be transformed into asynchronous processes. The CTT notations lack the support of an asynchronous modal of message exchange between tasks. Furthermore, the IFML also has "ActionEvent" constructs which are attached to an Action element and are produced by an Action to signal the termination of an operation. In other words, the Action construct waits for a response for the actual execution of an operation until it is able to modify the state of the UI as a result of a normal or exceptional termination. However, in IFML, the execution of an action produces an action completion event and the sending of an asynchronous notification which is matched by a "SystemEvent" construct. The cause of such a system event may be left unspecified in a model, denoting the absence of an actual message exchange. Thus, both CTT and IFML modeling elements provide support for a synchronous message exchange format.

- *Interruptibility*: BPMN and AD constructs can capture arrivals and conditions that can trigger an interrupt while particular steps are occurring-with BPMN providing a compact notation for doing this during a single activity with boundary events and an Activity diagram offering an inter-

rupt region with a dashed rectangle, both illustrated in figure 20. In this example, if an interrupt reaches due to the triggering of the Cancel Button or the time defined in the timer elapses, the "normal" sequence flow is disabled and the process continues in any of the sequence flows causing the interrupt. In the AD, the process continues along the interrupt flow connector (lightning symbol) to the outside also disabling all other activities within the region. In CTT, the "Disabling" temporal operator

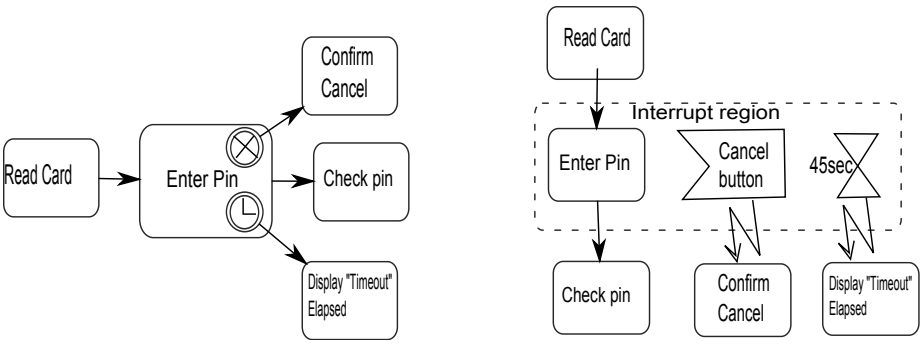

Figure 20: Example of interruptibility in BPMN (left) and AD (right)

provides semantics equivalent to the BPMN model and the AD discussed above. An example is when a user can iteratively input data in a form until the form is sent. Thus, the first task (usually an iterative task) is completely interrupted by the second task. The IFML also provides similar semantics with the help of system event constructs which can be timer events (triggered after an elapsed time-frame), system alerts (such as a database connection loss), or message receipt notifications. Nonetheless, these system events can be of an interrupt or a non-interrupt type. The later usually occurs during message notification system events which do not disable the current state of the UI and simply execute in parallel.

4.3. Informational Aspect

Data is an integral part of a workflow process. Hence, this section compares the four formalisms for the support of the four characteristics of data that occur repeatedly in different workflow modeling paradigms.

- *Data Visibility*: In BPMN and Activity Diagrams data visibility is mainly implemented by so called Pools/Lanes and Swimlanes respectively. These elements provide the possibility to partition a set of activities and their data elements from one element to the other and serve as organization units which deal with enfolding parts of a workflow processes. Aside from a more detailed view, BPMN and AD provide sub-processes and sub-activities respectively. These elements hide, in the collapsed state, all internal process definitions and processed data from other tasks.

In CTT notations, for each single task, it is possible to directly specify a number of attributes and related information. These data aspects can be classified in to three; namely *General Information:* where it includes the task identifier and its' extended name, its category and type, frequency of use, some informal annotation that the designer may want to store, indication of possible preconditions and whether it is an iterative, optional or connection task; *Objects:* where for each task, it is possible to indicate the objects (name and class) that have to be manipulated to perform it. Objects can be either user interface or domain application objects; *Time Information:* where it is also possible to indicate estimated time of performance; i.e. including a distinction among minimal, maximal and average time of performance [10].
IFML's modeling element, ViewComponent, publish data content in the interface. Hence, data visibility is mainly implemented by this construct in IFML. In addition, the source of the published content can be represented by means of the ContentBinding specification which is refined in two specializations, i.e. Data biding and dynamic behavior. A data biding represents the provenance of content from objects of a domain model concept such as the UML class diagram. It is also characterized by features that specify the type of data, the criterion for selecting instances, and the attributes of the object relevant for publication. Moreover, a dynamic behavior represents the data access of a ViewComponent in an operational way, e.g. through the invocation of a service or method that returns content.

- *Data Interaction and Transfer*: As data interaction and data transfer are tightly coupled, a summary of these two aspects is discussed among the formalisms. In BPMN, the previous mentioned pools support message flow for defining data interaction. The data transfer between defined elements within the workflow is realized well in BPMN via several types of 'data' objects such as data objects, data inputs, data outputs and data stores. There is also a well-defined way to manage the states of data, like instantiation, completed, deleted, etc. Likewise, to model data interaction in Activity Diagrams, it has a notation called action-object flow relationship, and includes two types of notation symbols- *object flow* and *object in state*. An object flow is the same thing as a transition line, shown as a dashed line in figure 21, and the object in state is the rectangle in the same figure. The underlined text in the rectangle is the object's class name and this class could possibly be found on one of the modeled system's class diagrams. The second part of the object in state is the text inside the brackets, which is the object's state name. CTT notations also support for data interaction, for instance, during concurrent communicating tasks; where tasks exchange information while performed concurrently. In addition, there are two general kinds of data objects in task models that actually interact: the *presentation objects*, those composing the user interface, and *application objects*, derived from the domain analysis and responsible

Figure 21: Action-object flow relationship

for the representation of persistent information. However, unlike BPMN and ADs, there is no clear modeling construct in CTT notations which specify visually the above mentioned data objects and the interaction pattern during the transfer of data between tasks.
IFML notation has a rich set of mechanisms to support data interaction and transfer. This is expressed through an input-output dependency between ViewComponents described by means of the ParameterBinding construct. A ParameterBinding specifies that the value of a parameter, typically the output of some ViewComponent, is associated with that of another parameter, typically the input of another ViewComponent. When the input-output dependency involves several parameters at the same time, ParameterBinding elements are grouped into a ParameterBindingGroup, as shown in figure 22.

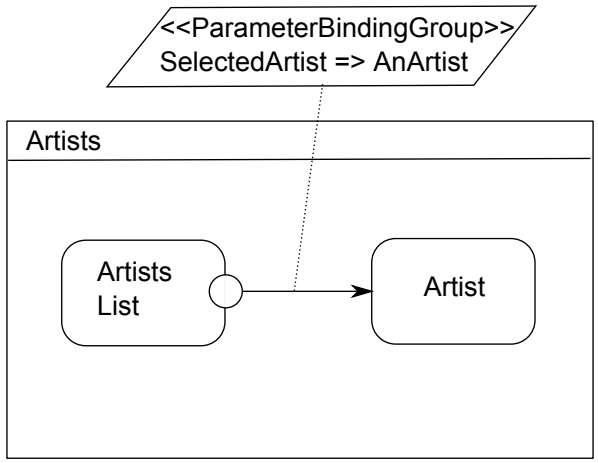

Figure 22: Example of input-output dependency in IFML

- *Data Based Routing*: This last data flow aspect deals with the manner in which data elements can influence the operation of other aspects of the workflow, particularly the control flow perspective. BPMN and AD make use of different kinds of constructs, such as gateways/nodes, to enable routing in the process definition. These elements generally provide

OR/XOR/AND and more complex decision based functions. Important for this aspect is the fact that these elements also accept data as a decision basis, hence making possible data-based routing.
Temporal operators in ConcurTaskTrees, such as *Enabling with information passing*, provides similar design pattern where the second task cannot be performed until first task is performed, and that information produced in first task is used as input for the second one.
In IFML, data based routing can be made possible with the use of ActivationExpression construct. A ViewComponent is active when the enclosing ViewContainer is visible and the values of its input parameters are available. Hence, ActivationExpression adds restrictions on top of the set of visible and active ViewComponents. If they evaluate to false, an otherwise visible/active element is treated as invisible/inactive, thus indicating a change in state of the user interface. For example, in figure 23, the toolbar with attached events for deleting, archiving, reporting, and moving messages remains inactive until at least one message is selected in the "MessageList" ViewComponent, as depicted by the MessageSet-notEmpty() ActivationExpression.

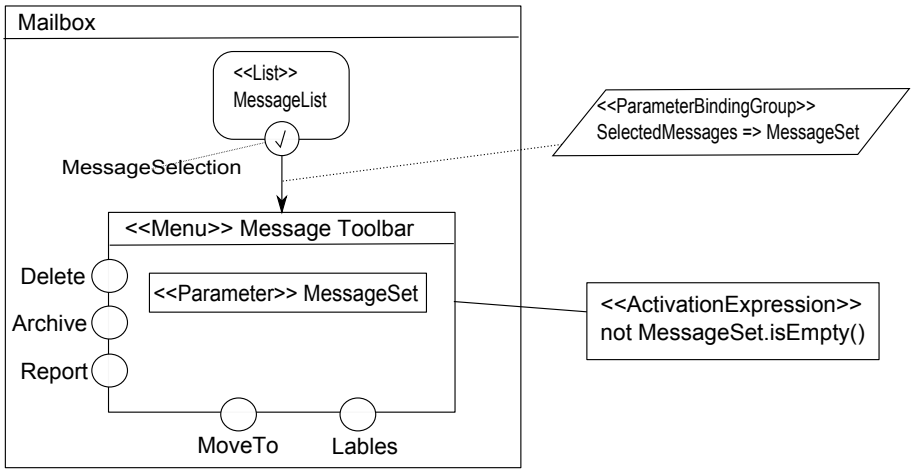

Figure 23: Example of data-based routing in IFML

4.4. Level of Abstraction

Both high and low-levels of abstraction represent a different model of the same information and processes, but a high-level description is one that is more abstracted, describes overall goals and systemic features, and is typically more concerned with the system as a whole, or larger components of it. While, a low-level description is one that provides a granular representation, rudimentary functions rather than complex overall ones, and is typically more concerned with individual components within the system and how they operate

at the implementation level.
In models with both, workflow and interaction flow patterns, it is desirable to create models with high level of abstraction in support of model analysis for dealing with system complexity. Thus, BPMN is suitable to use at a higher level of systems design to describe workflows at the level of business information but in the interest of describing BPMN models with high-level specification such as the inclusion of "Sub-process" constructs, a separate BPMN model can capture the details of a given sub-process and provide a higher-level of precision. BPMN also covers such a wide range of usage that it will map to more than one lower-level executable modeling language. As said above, BPEL is the primary language that BPMN will map to.
On the other hand, Activity Diagrams are commonly applicable to model business process or flow of work between users and systems at a higher-level of abstraction; for modeling steps performed in a use case; and for providing a model description of low-level methods/functions or operations in software systems. Similar to BPMN, ADs also describe the detailed behavior of sub-activities with "Call Behavior Actions" in support of modeling at a lower-level of abstraction. A called behavior is an activity diagram that is represented on the main activity diagram by a Call Behavior Action as illustrated in the figure 6. In addition, Activity diagrams also support for executability by mapping on to class diagrams as discussed in [13]. In the paper, Activity Diagrams are used in the modeling and implementation of graphical user interfaces, more precisely in the controller part of the Model-View-Controller pattern and provide a way to translate activity diagrams to class diagrams, which is an easily executable modeling language.
Task models can also be represented at various abstraction levels. When designers want to specify only requirements regarding how activities should be performed, they consider only the main high-level tasks. On the other hand, when designers aim to provide precise design indications then the activities are represented at a small granularity, thus including aspects related to the dialogue model of a user interface (which defines how system and user actions can be sequenced) [10]. Furthermore, CTT tasks have attributes such as the *Platform Attribute*, which allows designers to specify for what type of platform the task is suitable. It is also possible to specify for each object manipulated by the task in which platform it is suitable to support them. This type of information allows designers to better address the design of nomadic applications that can be accessed through a wide variety of platforms. In addition, once the task model has been developed, TERESA [14] is used for the user interface code generation. TERESA is a transformation-based tool that semi-automatically generates multi-device interfaces.
The IFML uses its core constructs to represent front end modeling at a higher level of abstraction encapsulating technological aspects such as executability on multiple platforms. However, despite IFML being a platform independent language, it has been designed in a way it can be executable. This is obtained through model transformations and code generation of IFML models constructed with its extension elements which add more expressive power to

technology specifications such as web, mobile and desktop platforms. Moreover, much like the previously discussed formalisms, IFML enables for specifying its modeling elements at different levels of precision. For example, at the most abstract level, a ViewComponent can be just a "box with a name" without further details which keeps the specification very general and easy to produce but may overlook important information needed for model checking and code generation. Thereby, at an intermediate level of abstraction, IFML allows a standard way of binding ViewComponent to elements of the domain model. This is important because, for example, a ViewComponent "Index of Products" can actually derive its content from the instance of a "Product" entity of the domain model. Furthermore, at the most refined level, the ViewComponent construct can be extended with specialized subclass to express specific ways in which content is presented. For example, a *List* ViewComponent can be defined in a way that aims at publishing an ordered set of objects from which the user can select one item.

4.5. Tools Support

BPMN has been around for quite long since its development and commercial usage. Hence, there are various open source and commerical tools in support of many features as shown in figure 24. We can also easly conclude that BPMN has rich documentation and community support mainly due to its standardization by the OMG and the many tools in support of BPMN integration for enterprise system modeling. Being part of the Unified Modeling Language

Name	Creator	Platform / OS	BPMN Version	Features
Activiti Modeler	Alfresco and the Activiti community	Cross-platform	BPMN 2.0	Modeler, Simulation, Execution
ActiveVOS	Informatica	Windows, Linux	BPMN 2.0	Modeling, Testing and Execution with open standards.
ADONIS (software)	BOC Information Technologies Consulting AG	Windows	BPMN 2.0	Business Process Analysis (BPA) tool supporting business process management allowing process modeling, analysis, simulation, evaluation, publishing and automation. Freeware Community Edition available.
Agiles BPMS & ECM	IMAGE Technology S.A.	Windows, Linux, Mac	BPMN 2.0	Modeler, Execution,
Altova UModel	Altova	Windows	BPMN 1.1, 2.0	Includes BPMN, UML, SysML, C# and Java round trip code generation, documentation, collaboration (including with MetaTeam) and database modeling

Figure 24: BPMN modeling tools and features. Image adopted from Wikipedia.

and a standard of OMG, Activity diagrams also have a rich set of tools and active community involvment. One particular tool in support of UML ADs is the Enterprise Architect, which enables for code generation from Activity diagrams in a Class that require a validation phase, during which Enterprise Architect

uses the system engineering graph optimizer to analyze the diagram and render it into various constructs from which code can be generated.
While task modeling and task-based design are entering into current practice in the design of interactive software applications, there is still a lack of tools and community involvment supporting the development and analysis of task models. However, the ConcurTaskTrees Environment (CTTE) [15] is an environment for editing and analysing of task models useful to support design of interactive applications starting with the human activities to support. Once the task model has been developed, TERESA [14] is used for the user interface code generation. TERESA is a transformation-based tool that semi-automatically generates multi-device interfaces. This tool uses the XML file, generated from the task model specified by ConcurTaskTree language. This XML specification is automatically generated by CTTE.
Moreover,IFML model analysis and code generation process is facilitated with the help of a commerical tool such as WebRatio and an open source Eclipse plug-in. However, these tools lack some essential features such as integrating IFML models to a certain behavioral modeling languages, e.g. UML class diagram, for specifying behavioral/operational aspects of a user interface. Both tools incoorporate Entity Relationship (ER) diagrams as a domain model from which IFML elements can refer to as a source of informational content display. WebRatio, on one hand provides external service components for invoking back-end operational aspects related to the application business logic. In addition, similar to CTT, IFML also has less community support mainly due to its recent development/launch and its late standardization by the OMG.

4.6. Integration with other formalisms

As previously discussed, many modeling formalisms are specialized to represent certain domains and hence do not allow for a comprehensive system design. In addition, they also enable to express models with high-level of abstraction which prohibit simple code generation mechanism. The focus of this section is to assess the possibility of integrating the previously discussed formalisms, with great emphasis to IFML, among one another and with a standard software engineering modeling paradigm, such as the UML, to support for an end to end system design and integration with executable languages for code generation. In light of this, IFML, BPMN, and AD are well adopted by OMG's Model-driven architecture (MDA) framework, which is a software design approach for the development of software systems, and supports model-driven engineering (MDE) of software systems. Thus, being part of the MDA framework enables for a tight integration of each formalism with other system modeling perspectives. Hence, we will first look at how IFML, BPMN, and AD integrate with each other and with other UML modeling languages to achieve the aforementioned system design goals.
In many cases, system development starts from a requirements model, such as UML use case diagrams, or from a procedural view of the enterprise operations, such as business process models specified in OMG's BPMN [1]. IFML enables the traceability of user interaction models to the requirement specifications that

generated them; made possible by establishing a reference between the requirements model of interest, such as BPMN, and the IFML model derived from it, supported by the tool WebRatio. Moreover, the internal functioning of an IFML Action construct, which is considered as a "black box" for being unaware of internal state changes of a system, can be specified by referencing a method in a UML class diagram. If the action is described by a complex behavior, this can be obtained by referencing a UML dynamic diagram such as sequence diagram or activity diagram. However, there is a lack of tool incoorporating this. In addition,in WebRatio, the IFML ViewComponent construct specifies the publication of it's content through ContentBinding, which establishes a reference between the IFML diagram and a domain model ER diagram, where the object of interest attributes are defined.
Furthermore, aiming at integrating task models represented in ConcurTaskTrees and the UML is of great interest mainly because not all UML notations are equally relevant to the design of interactive systems. Task modelling phase of interactive system development allows designers to obtain an integrated view of functional and interactional aspects. Interactional aspects, related to the ways to access system functionality, in particular cannot be captured well in use cases; which, however, are commonly used during the requirement elicitation phase. Moreover, in system's design, there is a desire to achieve a complete identification of the objects belonging to the domain considered and the relationships among them. Designers need to associate tasks with objects in order to indicate what objects should be manipulated to perform each task. This information can be directly introduced in the task model. In CTT it is possible to specify the references between tasks and objects. However, in the domain model, more elaborate relationships among the objects are identified, such as association, dependency, flow, generalisation, etc., and they can be easily supported by UML class diagrams.
Another aspect of integrating formalisms is to facilitate an easy mechanism for code generation. As discussed in previous sections, BPMN can be mapped onto BPEL, a low-level executable language. In addition, Activity diagrams can be translated to class diagrams for executable code. Using model transformation techniques, CTT task models can also be mapped to TERESA for user interface generation, as discussed in tools support section. IFML is also designed in a way it can be executable through model transformations and code generation of its models constructed with its extension elements which add more expressive power to technology specifications such as web, mobile and desktop platforms.

5. Experimental Analysis

The comparative analysis can be well supported with an experimental demonstration as discussed in this section. This experiment part consists of three tasks.

- *IFML model representation for a BPMN model and an Activity Diagram*: This experiment deals with constructing an "Online Shopping Applica-

tion" which allows the three formalisms to model the application in different perspectives as follows:
A **BPMN Model** is used as a process model to describe the work flow of the application at the level of business information. Hence, When the user enters into the website, starts exploring the products available. Once he finds a product of interest, selects it, and the item goes to the shopping cart. The user can either keep exploring products in order to add more items to his order, or continue to manage the shopping cart by deleting all the products, or updating quantities of the selected ones. Once the user is ready to proceed with the payment, performs the checkout. In order to authorize the payment, it's necessary to send the customer information to the bank entity, and wait for the confirmation. This procedure is illustrated in figure 25. At a more refined level, the **Activity diagram**

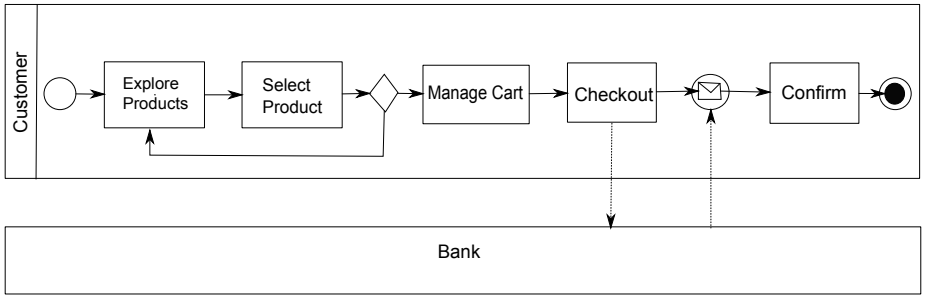

Figure 25: BPMN model for Online Shopping Application

with data flow illustrated in figure 26, is used to model the application describing the steps undertaken in a corresponding use-case and denotes the procedural flow of actions, including data flow, assumed to be fulfilled by the online customer.

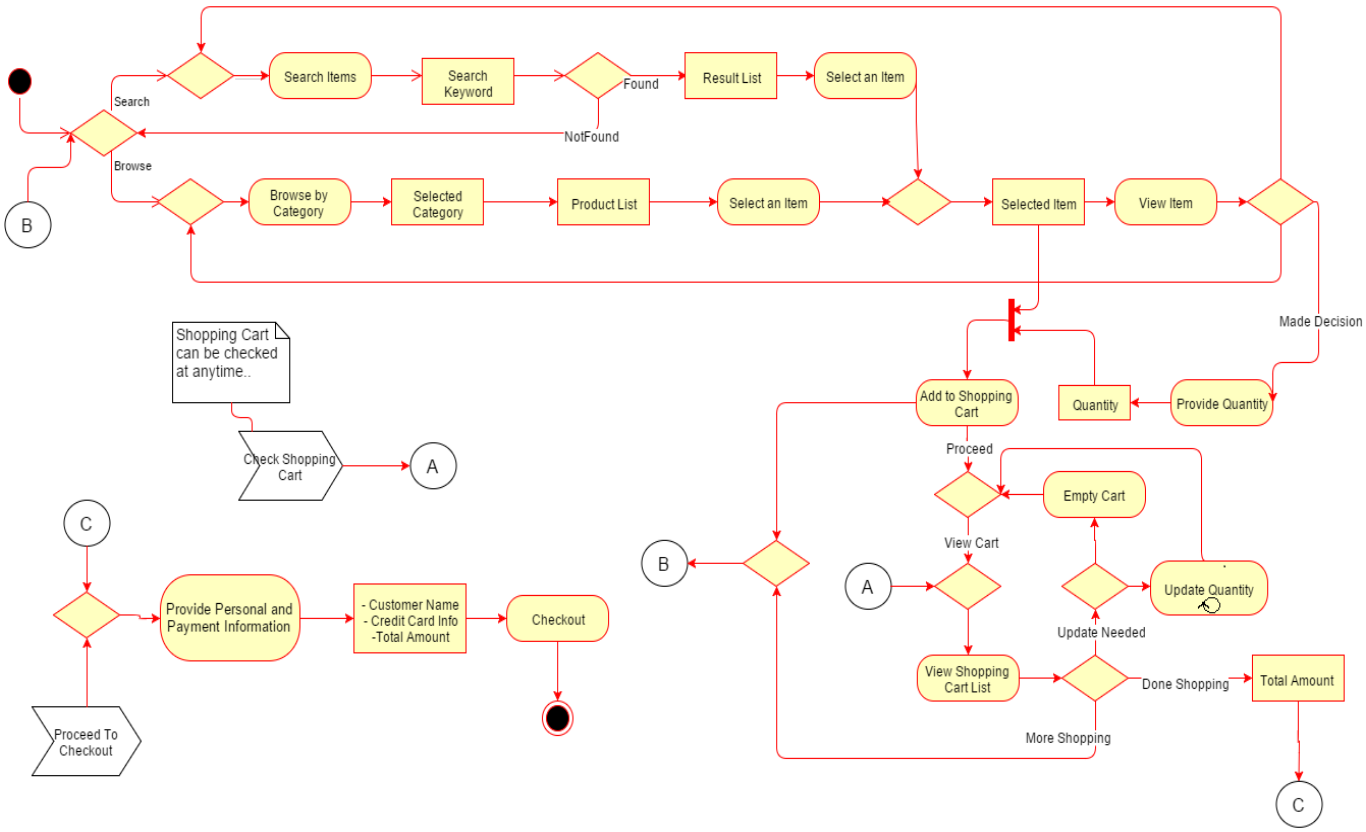

Figure 26: Activity diagram for Online Shopping Application

The **IFML model** typically illustrates the front-end modeling of the application corresponding to the two models described above. Figure 27 shows that the user's interaction flow at the application level actually comply with the activity diagram denoting the procedural flow of activities undertaken. This can be further illustrated in the results and discussion section. Thus, the purpose of this experiment is to deduce if the Activity

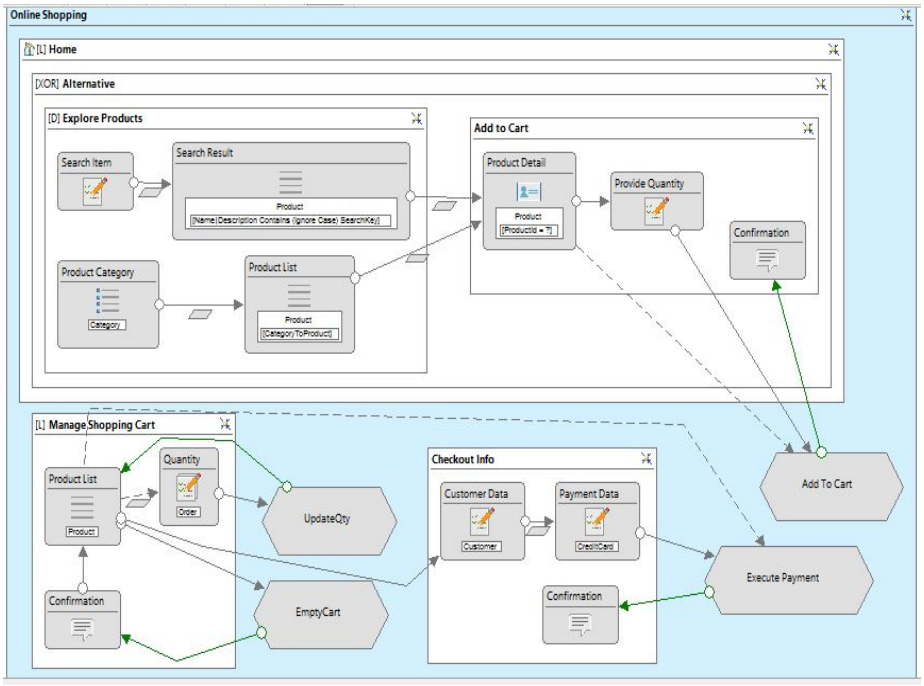

Figure 27: IFML model for Online Shopping Application

diagram constructs, from the control-flow and data-flow perspective, can be emulated in IFML with *exactly* the same semantics.

- *"Crazy Rectangles" demo in IFML*: The focus of this experiment is on UI modeling utilizing the IFML formalism. The model incorporates special features such as the support for *timed events*, triggering and handling events *autonomously*, *reactive* to events from outside, e.g. user, and possible to instantiate *multiple objects dynamically*.
  Hence, the goal of this experiment is to model an application that when started, spawns a window. If a button is clicked in the window, a rectangle object which can contain text labels, is created at that location, and it moves around in the window canvas but bounces back from the sides of the canvas. It is also possible to create multiple rectangles in the window denoting multiple instantiation and dynamic structure of the application.

Additionally, there is a button to create new windows, exactly like the first one, where it is also possible to create new rectangles. Key presses also trigger event handlers to, for example, edit a text label, rotate, scale, move objects, create new objects, etc.

- *Code Generation*: As the focus of this paper is on executable models following the MDE paradigm, it is imperative to assess how code can be generated for the above discussed two modeling tasks.

5.1. Results and Discussion

This section provides an elaboration to the results obtained while conducting the above mentioned experimental tasks.

5.1.1. IFML model representation for a BPMN model and an Activity Diagram

As mentioned previously, the purpose of this experiment is to deduce if the Activity diagram constructs, from a control-flow and data-flow perspective, can be emulated in IFML with *exactly* the same semantics. Hence, a subset of both formalisms is selected to illustrate model transformation is possible, as follows:

- *Merge Node and Exclusive Join in IFML*: In the IFML model, the user can choose to view a particular item detail by triggering either one of the events attached to the "Search result" or "Product List" ViewComponent, which pass the selected item object to the "Product detail" ViewComponent to make it visible the its parent "Add to cart" ViewContainer active. Similarly, this pattern is observed in the AD by the "Merge" node preceded by two "Select an Item" action constructs where the execution of either one of them will change its procedural flow to yield a "Selected Item" data object as an input to "View Item" action.

- *Parameter Binding and Object Nodes*: A "ParameterBinding" construct is in place for input-output data dependency between any IFML elements that capture data-flow in the IFML model. It is denoted by the small rectangular shapes associated to a navigational flow indicating the data dependency among the source and the target element. Also to note is that the IFML "ViewComponent" constructs publish their data content from a referenced domain model source, i.e. ER diagram as in Figure 28. Likewise, the AD makes use of "Object Nodes" as the simplest method of describing the information flowing between activities. However, another way of capturing data flow in AD, which uses an "Output Pin" and an "Input Pin" constructs, provides a more precise semantics equivalent to the IFML "ParameterBinding" constructs since it enables to separately describe the outputs from one action and the inputs to another.

- *Join Node and Synchronization in IFML*: The "Add to cart" Action construct in the IFML model is triggered as a result of the synchornization of both input flows directed to it. The data flow denoted by a dashed line

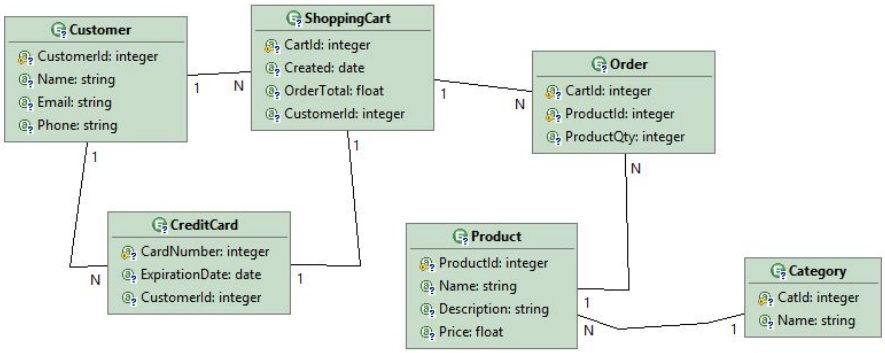

Figure 28: ER diagram as a domain modeling concept for the Online Shopping Application

is enabled when the attached "Product detail" ViewComponent is active and passes the product object to the action. However, the action construct is triggered when the other input navigational flow is enabled as a result of providing quantity by the user and the quantity data is binded with the action. In a similar manner, the AD has a "Join" node which takes two input and produces an output action "Add to shopping cart". This action is triggered as a result of enabling both input Object Nodes, i.e. "Selected Item" and "Quantity", thus representing synchonization control-flow pattern. It is worth to note that synchonization in AD has a slightly different semantic than IFML because it implies that the input branches for the "Join" node actually perform in parallel while in IFML it can be in any order.

- *Iteration*: When the state of the interface changes to "Manage shopping cart" ViewContainer in the IFML model, it is possible to repetedly provide update to the quantity of an item. This is because everytime when the "UpdateQty" action is triggered and it successfully executes, it preserves the state of the interface which remains unchanged but the sequence of interaction flow carried out to perform an update can repeat until the user wishes not to. This control flow pattern is also easily recognized in the AD denoted by the "Update Quantity Iteration" node.

- *Decision Node and Conditional branching in IFMl* : In the IFML model, when the state of the interface is in "Manage shopping cart" ViewContainer, the user can choose to do more shopping by accessing the home page or proceed to checkout by triggering the corresponding attached event. The "Home" ViewContainer is accessable because it has a steriotype [L] denoting the page is accessible by its siblings and their children. Similarly, the AD makes use of "Decision" nodes to provide for the same control-flow pattern where it has an input branch from "View shopping

cart list" and three possible conditional branchings as an output.

5.1.2. "Crazy Rectangles" demo in IFML

For modeling the desired behavior of this demo application, IFML supports for "Event" constructs that can be produced by a user's interaction, by the application itself, or by an external system. Moreover, these modeling elements also yield event transitions that specify the consequence of an event on the UI. These transition can be:

- A change of the ViewContainer (e.g a web page),
- An update on the content on display (e.g. data published on ViewComponents placed on a ViewContainer),
- The triggering of an "Action" construct, or
- A mixture of these effects.

Hence, the IFML "Action" construct has such a semantics denoting the execution of these types of operations (e.g. services performed by the application business logic). However, the "Action" construct in IFML is considered as a "black box" for not describing the internal functioning of an operation. The goal of IFML's action construct is rather to express the interplay (i.e. the minimal amount of information needed) between the interface and the business logic. This is done by:

- Showing that an event triggers an operation (business action), which may imply also the specification of some input-output dependency between the information carried by the interface and the business logic, and/or
- Describing the input-output dependency between the information carried by a system event and the affected elements of the interface, in the case when the interface receives and responds to events generated by "the system".

Thus, IFML does not replace the behavioral specifications that are normally employed to describe the algorithmic aspects of an operation. In support of such refinement, an Action construct in IFML can reference a behavioral modeling language such as the UML class diagram, which are not only containers of information but also allow the specification of behavior. IFML refers to class objects that provide data content to be published in the application front-end; and events triggered within the interface may cause the execution of class methods, which may update objects and change the status of the interface.
However, the two tools available in support of IFML, at the time of publication, (i.e. WebRatio and IFML plug-in on Eclipse environment) allow for referencing a domain modeling concept- i.e. the Entity-Relationship model, only to describe what pieces of data are published in the interface. Hence, the fact that IFML, on its own, has a limitation for not describing the internal behavioral specifications of an object, and the lack of tools support for referencing an IFML model

to a behavioral modeling language has been a major obstacle to further pursue the task illustrated in the "Crazy Rectangles" experiment.

5.1.3. Code Generation

This section provides the results obtained while executing the IFML model capturing the UI components and their interaction flow represented in an "Online Shopping Application". Thus, the WebRatio community platform, being the prefered tool for this task, has the option of generating and deploying IFML models on cloud. However, the "Action" constructs generated an error mainly due to lacking an external service call to execute the specified operations. This is because the IFML model representing the front-end of the application has to create and connect with a certain database system utilizing the available entities in the corresponding ER diagram and specify the operations as an external service, such as "Add to Cart", to actually create objects in the connected DBMS. Thus, generating a fully fledged web application will be out of the scope of this paper's work and requires a learning curve that deal with other technologies integrated with the IFML model in the working platform.

6. Conclusion

In this paper, a literature review has been conducted for the task of UI modeling in the context of a model-driven engineering paradigm. The IFML formalism, an OMG standard for front-end design, has been extensively explored for its appropriateness by assessing its support for essential features indicated in workflow design patterns. In particular, a set of evaluation criteria has been drawn for comparing the IFML with similar formalisms, such as BPMN, AD, and CTT, in addition to conducting relevant experimental tasks for further assessment. The subsequent findings are also presented.
Thus, after analyzing the results of this comparative study, we can conclude that the following areas stand to be uncommon in the literature review and the experimental tasks, hence relevant for future reserach in UI modeling:

- In IFML, complex user interaction pattern, such as multiple instantiation of an object at run-time, is not discussed in the literature reviewed and thus indicates its lack of support.
- The IFML notations support for Asynchronous message exchange (service invocation) modality is not explored in the literature reviewed.
- The IFML's "Action" construct lacks the appropriate semantics which supposedly denotes the purpose of expressing and executing behavioral aspects of a user interface. Instead, it simply serves as an interplay for binding information between the front-end and the business logic of an application.
- In support of further refinement to the IFML's "Action" construct, it is indicated in the literature reviewed that it can reference behavioral

modeling languages, such as the UML Activity diagram, for specifying its internal operational aspects which dynamically influence the state of UI components. However, the limited number of tools supporting the IFML formalism do not implement this essential feature.

The IFML, however, addresses many of the design patterns incoorporated in workflow modeling languages, particularly of an interest are the control-flow and data-flow perspectives of an interactive systems modeling. Nonetheless, in light of addressing the aforementioned issues to identify an appropriate formalism for a model-driven UI engineering task, a more pragmatic approach with the appropriate tools support for multi-formalisms, such as a subset of the IFML with other behavioral modeling languages, is proposed.

References

[1] M. Brambilla, P. Fraternali, Interaction Flow Modeling Language: Model-Driven UI Engineering of Web and Mobile Apps with IFML, Morgan Kaufmann, December 3, 2014.

[2] S. A. White, Process modeling notations and workflow patterns.

[3] Object Management Group, http://www.omg.org/spec/BPMN, Business Process Model and Notation (BPMN), bPMNv2.0.2 Specification Document (12 2013).

[4] R. M. Dijkman, M. Dumas, C. Ouyang, Semantics and analysis of business process models in BPMN, Information and Software Technology 50 (12) (2008) 1281–1294. doi:http://eprints.qut.edu.au/7115/.

[5] W. P. van der Aalst, Business process management: A comprehensive survey, ISRN Software Engineering 2013 (2012) 37. doi:w.m.p.v.d.aalst@tue.nl.

[6] S. A. White, Introduction to BPMN, Tech. rep., IBM Corporation.

[7] P. Wohed, M. Dumas, A. T. Hofstede, N. Russell, Pattern-based analysis of uml activity diagrams, in: Eindhoven University of Technology, Springer, 2004, p. 242.

[8] H. Strrle, Semantics and verification of data flow in uml 2.0 activities, in: In Electronic Notes in Theoretical Computer Science. Elsevier Science Inc, Elsevier, 2004, pp. 35–52.

[9] D. Bell, Uml basics- the activity diagram, Tech. rep. (2004).

[10] F. Patern, Concurtasktrees: An engineered approach to model-based design of interactive systems (2000).

[11] M. Vasko, S. Dustdar, A view based analysis of workflow modeling languages.

[12] N. Russell, A. T. Hofstede, D. Edmond, Workflow data patterns, Tech. rep. (2004).

[13] J.-P. Barros, L. Gomes, From activity diagrams to class diagrams.

[14] S. Berti, F. Correani, G. Mori, F. Patern, C. Santoro, Teresa: A transformation-based environment for designing and developing multi-device interfaces.

[15] G. Mori, F. Paterno, C. Santoro, CTTE: Support for developing and analyzing task models for interactive system design, IEEE Transactions on Software Engineering 28 (8).