Model Driven Development of Groupware Systems

Luis Mariano Bibbo
LIFIA - Facultad de Informática - Universidad Nacional de La Plata, Argentina

Claudia F. Pons
Centro de Altos Estudios en Tecnología Informática (CAETI) de la Universidad Abierta Interamericana (UAI), Buenos Aires, Argentina
CICPBA, Comisión de Investigaciones Científicas, Buenos Aires, Argentina
LIFIA - Facultad de Informática - Universidad Nacional de La Plata, Argentina

Roxana Giandini
Grupo GIDAS - Universidad Tecnológica Nacional - Facultad Regional La Plata
CICPBA, Comisión de Investigaciones Científicas, Buenos Aires, Argentina
LIFIA - Facultad de Informática - Universidad Nacional de La Plata, Argentina

ABSTRACT

Building Collaborative systems with awareness (or groupware) is a very complex task. This article presents the use of the domain specific language CSSL v2.0 - Collaborative Software System Language - built as an extension of UML, using the metamodeling mechanism. CSSL provides simplicity, expressiveness and precision to model the main concepts of collaborative systems, especially collaborative processes, protocols and awareness. The CSSL concrete syntax is defined via a set of editors through which collaborative systems models are created. According to the MDD methodology, models are independent of the implementation platform and are formally prepared to be transformed. The target of the transformation is a web application that provides a set of basic functions that developers can refine to complete the development of the collaborative system. Finally, evaluation, validation and verification of the language is performed, determining that the CSSL tools allow developers to solve central aspects of collaborative systems implementation in a simple and reasonable way.

Keywords: Groupware, Collaborative Software, Awareness, Model-Driven Engineering, Meta-Model, Code Generation

INTRODUCTION

Groupware also known as Collaborative Systems are applications in which a group of users, following a common goal, develop different joint activities. The collaborative software allows them to share information, communicate, collaborate, and coordinate joint activities. According to (Ellis, Gibbs, and

Rein 1991; Grudin 1994), collaboration platforms are “Computer-based systems that support groups of people engaged in a common task (or goal) and that provide an interface to a shared environment”.

A main characteristic of the collaborative systems is the equilibrium between the individual work and the effort the users must do to achieve the common goal. To this end, the collaborative system offers coordination mechanisms. (Dourish and Bellotti 1992; Grudin and Poltrock 1997) for organizing the activities. For example, a coordination protocol can specify the collaborative tasks to be developed, the order in which they must be developed. and the specific tasks that each role can perform in each state at any time.

Additionally, to take more advantage of the shared environment, the users can be informed regarding the actions that the other users perform and how these actions affect the work environment (Gutwin and Greenberg 2002; Gutwin, Greenberg, and Roseman 1996). This information provided by the system is known as awareness. According to (Dourish and Bly 1992) the awareness is the perception or knowledge of the group and of the activities performed by others that provides context for your own activities. In particular, the awareness allows users to coordinate their work based on knowledge of what others are doing or have done. For example, users will be able to see the changes in the shared documents or the degree of progress of the common tasks, as explained in (Collazos et al. 2019).

The most frequent types of awareness are presence, location, density, user data (Age, Nationality, etc.), activity level, actions, places where you were, places where you performed the actions, changes you made, objects you control, objects you can reach, information you can see, intentions, abilities, influence, and historical data about the previous ones. Awareness information can also be grouped together to improve awareness effectiveness. For example, when the user's presence is displayed, other information such as status, location, or resources can be attached to it.

Without doubt, building collaborative systems with awareness is a very complex task that requires powerful tools. However, traditional approaches, based on mainly coding the applications, are still used in the development process of these software systems. On the one hand, there is no clear documentation of design decisions taken during the coding phase, making the evolution and the maintenance of the systems difficult. On the other hand, when using specific programming languages, the possibility of generalizing concepts - that could be extracted, re-used and applied in different systems - is wasted and the code is written from scratch over and over again.

In this context, the goal of our work was to investigate the application of modern technologies for developing collaborative systems, in particular the Model Driven software Development approach (MDD) (Stahl et al. 2006), that proposes to improve quality and efficiency of the software construction processes. In this paradigm models assume a leading role in the software development process, going from being contemplative entities to becoming productive entities from which implementations are automatically derived.

The MDD initiative promotes:

- Abstraction: the use of a higher level of abstraction in both the specification of the problem to be solved and the corresponding solution.

- Automation: increased confidence in computer-aided automation to support analysis, design, and execution.

- Standardization: the use of industrial standards to facilitate interaction between applications and technological evolution.

One of the key benefits of applying MDD is the flexibility to face technological changes. High-level models are free of implementation details, which facilitates adaptation to changes to the underlying technology platform or the deployment architecture.

A basic concept used in the MDD field is the idea of creating models for a specific domain through Domain-Specific Languages (DSLs), focused and specialized to that domain. These languages allow designers to specify the solution using problem domain concepts directly. End products are then automatically generated from these high-level specifications.

Following our goal of finding better tools for developing collaborative systems, we designed and implemented CSSL v2.0 -Collaborative Software System Language-, a DSL built using the metamodeling mechanism as a UML extension. This DSL has been validated and verified.

The rest of the paper is organized as follows. Section 2 describes the most relevant works related to the modeling of collaborative systems. Then. section 3 describes the abstract syntax of CSSL v2.0, using the metamodeling technique. Section 4 presents the semantics of the language by means of a model-to-code transformation. Section 5 elaborates a case study about the application of the language to model a complex collaborative system Section 6 compare CSSL with related works mentioned, Section 7 shows the evaluation of the DSL using as reference the work of (Robert et al. 2009) to later discuss its validation and verification using as reference the work (Sargent 2013). Finally, conclusions and lines of future work are presented.

RELATED WORKS

In this section we present a discussion of the most relevant works related to the modeling of collaborative systems with awareness. The information is an update of a systematic literature review (Bibbo, Giandini, and Pons 2016), where software engineering resources to design Collaborative Systems in a standardized, efficient and automated way were searched and analyzed.

Firstly, a cluster of documents published by a group of scientists from the Laboratory of User Interaction and Software Engineering of Castilla La Mancha University were analyzed. In these articles (Belkadi et al. 2013; Gallardo et al. 2011; Gallego et al. 2011; Kamoun, Tazi, and Drira 2012; Teruel et al. 2011, 2012, 2013, 2014; Vieira, Tedesco, and Salgado 2010), published between 2011 and 2014, the authors progressively define a language to model CSCW systems requirements. Initially, a language named CSRML (Teruel et al. 2011) (Collaborative System Requirements Modelling Language) was proposed as an extension of i* Goal-Oriented specification. The language presents a set of basic elements for modeling the special collaboration features of CSCW systems, such as goal, role, actor, task and awareness. These elements are connected via a set of relationships, for example Playing link, Participation link, Responsibility link. Then, in 2012 the authors conducted various experiments and comparisons with different Goal Oriented techniques (Teruel et al. 2012) (e.g., NFR framework, i* and Kaos) in order to determine which is the most suitable one to specify requirements of collaborative systems. They showed that the understandability was higher for the models specified with CSRML than for those specified with other methods, especially for collaborative aspect. Then they build a CASE tool that provides support for CSRML for specifying CSCW system requirements (Teruel et al. 2013). The tool was implemented as a Visual Studio 2012 extension by using the Visualization and Modeling SDK. Finally, in 2014 they made a usability experiment of the CSRML CASE tool 2012 (Teruel et al. 2014), identifying some usability flaw to be solved in the next releases of the application, as well as giving a general overview on how to develop this kind of tools by means of Visual Studio Modelling SDK.

The CSRML language is based on the abstract concepts of "elements" and "relationships", from which, concrete elements are defined, such as goal, task, role, resource and general awareness. This definition leaves out other important concepts such as Session, Tool and Workspace. The CSRML language is not expressive enough for the specification of collaborative processes seen as a set of ordered tasks in pursuit of a common goal. Several collaborative specifications, such as “which tasks can be performed at any time”, “which sessions can be performed at a given time”, or “under which conditions a task can be run”, cannot be expressed in the language. Because of this drawback, awareness information related to the processes cannot be defined in the model. For example, how advanced the processes are, how many of them are active or what process is about to begin.

Although a detailed description of the CSRML language features is provided, it fails to define a rigorous metamodel supporting the language. On the other hand, the CSRML CASE tool 2012, is not based on standards and is not integrated with the standard UML, which prevents us from using UML related tools such as plugins, code generators or editors.

In (Belkadi et al. 2013) a detailed literature review about the concept of awareness is presented. This review helps to identify key awareness-related requirements for the development of collaborative systems. Researchers points out that the awareness is a multi-faceted concept, and thus they have proposed different types of awareness and added adjectives to define the key facets of this concept (social, mutual, activity, etc.). They also concluded that collaborative systems which intend to support awareness must meet a large number of requirements. In (Gutwin et al. 1996), a list of questions covering the main facets of awareness was proposed.

Regarding this literature survey, the following concepts have been frequently used and should be the basis of a robust awareness-focused model:

- **Context element:** The context or situation is modeled through a set of elements
- **Task and activity:** It describes what is expected to be done and what the actor is really doing.
- **Resource:** It describes an element of the context. This element contributes to, or is used during, the fulfillment of an interaction.
- **Interaction:** The concept of interaction plays a central role in activity theory, in collaborative design and in Business Process Modeling (BPM).
- **Role:** According to the theories of organization, a generic structure for these interactions can be defined according to the major categories of contributions (or roles). In a normal work situation, functional roles are formally established as particular job positions.

In (Kamoun et al. 2012), FADYRCOS (A Framework for DYnamic Reconfiguration of networked COllaborative Systems) is presented. It supports semantic adaptation enabling the awareness of the presence/absence, roles and tasks of collaborators. This framework is based on a generic multi-level modeling approach that ensures multi-level adaptation. They intend to support collaboration in distributed environments where sessions can be implicit, new mechanisms are needed for managing session evolution and role changes. A multi-level modeling architecture is proposed with 3 levels:

- **Collaboration Level:** The collaboration level provides a session level abstraction. It describes how the members of a group are organized within sessions where they can send and receive data flows.
- **Application level:** The retained model in this level is a domain specific ontology that represents concepts and relations modeling the context of the application. Such ontology depends on the application domain, hence it will be provided by the designer of the collaborative system using

the framework. The main generic collaboration elements that will be specialized are: Node, Group, Role and hasSession.

- **Messaging Level:** For this level, two communication paradigms are considered: the Event Based Communication and the Peer-To-Peer communication.

One of the interesting features of the work is the deployment service where adaptation needs related to collaboration are handled. On the one hand, a set of services representing the current state of a session is proposed (Connect, Quit, AddToGroup, AddRole, RemoveRole, etc.) and on the other hand it allows the dynamic creation of spontaneous sessions between participants (createSession, closeSession, joinSession , quitSession). These services allow you to link the specifications made in the design stage with different implementations.

In (Gallardo et al. 2011) an ontology of awareness for modeling collaborative systems is presented. First of all, a review was performed with the aim of elaborating a theoretical foundation, which is useful for an adequate understanding of the work. One of the most outstanding contributions in this field is the Theory of Awareness by (Gutwin and Greenberg 2002), which includes a framework that defines different awareness elements and makes validation of awareness support possible by means of a set of relevant questions ("Who, What, Where").

The ontology presented is divided into three sub-ontologies. First, we have the sub-ontology of the application domain, which includes the application domain concepts, such as entities, properties and relationships. Next, the workspace sub-ontology, which deals with the collaborative tasks of the modeling process to be supported and the tools used to implement such tasks. Finally, a new sub-ontology was added, describing the concepts relating to awareness in the scope of collaborative systems for modeling.

The study addresses the awareness problem in collaborative systems in two aspects. On the one hand the concept of awareness appears modeled with two subclasses Workspace and Group Awareness. On the other hand, introduces the concept of Awareness mechanism depicting how the awareness is going to be handled. For example, when you want to inform that a user accesses to a session, the Access Awareness Mechanism is used. In other words, on one hand we specify the "What kind of awareness is reported" and on the other hand, we describe "Which event/action is activated".

In (Gallego et al. 2011) a set of Awareness Support Widgets is described. This proposal is an extension of a previous work of the same authors published in (Gallardo, Bravo, and Redondo 2012; Molina et al. 2013). In this metamodel the connection between the awareness model and the traditional concepts of collaborative systems (e.g., Workspace, Session, Collaborative Process, Tool, etc.) is not clearly defined. Additionally, certain types of awareness are not supported by the metamodel, for example awareness-related activities performed by users cannot be specified and the progress of a collaborative process cannot be displayed.

In (Vieira et al. 2010) a metamodel called Context Metamodel is presented. This work is not specifically about awareness; however, it includes some interesting concepts to specify the awareness in collaborative systems. The main class in the metamodel is ContextualElement class (like collaborative element used in groupware). Awareness information can be attached to this class. In the proposed metamodel there are a set of useful enumeration types, such as the ContextType that lists different kinds of awareness (i.e., the Gutwin's and Greenberg's list: "who", "what", "where" "when" "why"). Another enumeration type is the UpdateType defining when the awareness is updated. And finally, the AcquisitionType determining the mechanism by which awareness information is acquired.

Using the proposed metamodel collaborative processes cannot be fully modeled. Several collaborative aspects, such as "which tasks can be performed at any time", "which sessions can be performed at a given time”, or “under which conditions a task can be run”, cannot be expressed in the model. Because of this drawback, awareness information related to the processes cannot be defined in the model. For example, how advanced the processes are, how many of them are active or what processes are about to begin.

In (Briggs, Gert-Jan de Vreede, and Kolfschoten 2007) the idea of thinklet is presented as a design patterns for collaborative work practices. Collaboration engineers use thinkLets as building-blocks for creating reusable collaboration process designs to be transferred to practitioners to execute for themselves without the ongoing intervention of professional facilitators. The collection of thinkLets forms a pattern language for creating, documenting, communicating, and learning group process designs. Each thinkLet must specify a set of rules that prescribe the actions that people in different roles must perform using the capabilities provided to them under some set of constraints specified in the parameters.

In (Gallardo et al. 2012) a model-driven method for the development of domain-independent collaborative modeling tools was proposed. This method consists of a methodological framework, a conceptual framework and a technological framework. The methodological framework defines the phases to be carried out when applying the method, whilst the conceptual framework is made up of the meta-models used in the method and the transformation processes established between them. Finally, the technological framework consists of the integration of some plug-ins from the Eclipse Modeling Project with some add-ons which provide collaborative functionality. The work proposes a domain design within the conceptual framework, describing the atomic elements to be edited collaboratively. The collaborative tools that will be used in the system are some specific editors and a few other classic tools of collaborative systems (i.e, chat or news).

Finally, in (Molina et al. 2013) the main conceptual frameworks were analyzed and the concepts that are generally used are specified. (Activity, Action / Operation, Agent / User, Group, Role, Tool, Shared Object / Resource, Awareness Rules). Furthermore, it is mentioned that modeling cooperation involves the inclusion of special coordination tasks at the end of the cooperative activity to enable the group to collect their individual contributions in the final product (group solution), as well as decision-making or agreements in this production process. In this latter case, they are talking about the existence of protocols for interaction and coordination among the group members and they come to the following conclusion:

- The language should support the modeling of different levels of abstraction and, in particular, the decomposition of tasks into subtasks.
- It should be able to specify aspects of coordination (one of the basic concepts in CSCW systems).
- The models that specify the work to be done will allow specifying the workflow.

Molina’s work presents the abstract syntax of CIAN notation and provides a conceptual framework that encompasses the elements previously identified as relevant to model groupware applications. Like the work in (Gallardo J et al), a model describing the atomic elements to be edited collaboratively is supported; this may be useful in some domains, but it restricts the variety of potential tools to be used.

SYNTAX OF THE DOMAIN-SPECIFIC LANGUAGE

Like any languages CSSL has both an abstract and a concrete syntax. The abstract syntax specifies the language structure while the concrete syntax, which is usually a graphical notation, defines the way language constructions look like to the user. It is important to note that the same abstract syntax might have several different concrete syntaxes. Metamodeling is a well-established and effective technology in the field of language engineering. In the case of the CSSL language, this approach is used to express the abstract syntax while a set of graphical editors are applied for implementing the concrete syntax. These editors allow metamodel instantiation.

Abstract Syntax

The CSSL metamodel was built as an extension of the UML metamodel. Therefore, it includes UML and all its metaclasses can be used. To give an example, Figure 1, Figure 2 and Figure 3 show in dark gray some UML metaclasses with their CSSL-specific extensions.

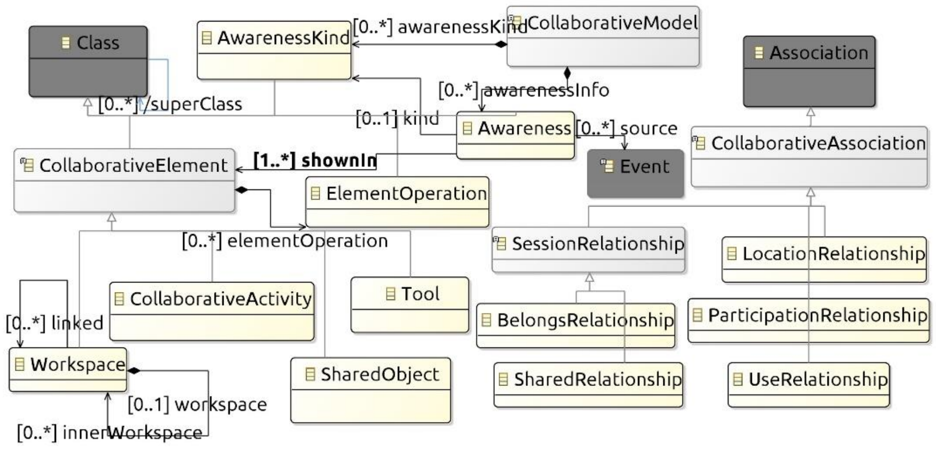


Figure 1: CSSL Conceptual Model

Figure 1 shows the main metaclasses of the CSSL metamodel: CollaborativeElement which is a subclass of UML Class, and CollaborativeAssociation which is a subclass of UML Association. This language allows users to work with the main concepts of collaborative systems: "Shared Object", "Tool", "Collaborative Activity", "Workspace", "User", "Role" and connect them to express the characteristics of the system. For example, an activity involves certain roles and uses some tools (elements are specified and related). In addition, the language allows users to describe the awareness elements that appear in the collaborative system and how it relates to the elements being modeled. This is reflected in the metaclasses Awareness and AwarenessKind that can be connected to any collaborative element by means of the shownIn relationship.

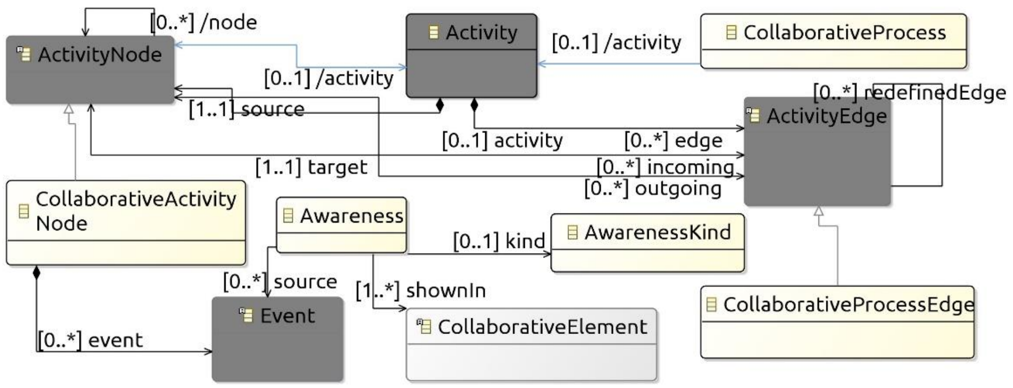


Figure 2: Collaborative Processes in CSSL

Figure 2 shows that CollaborativeProcess behavior is represented through UML Activity that is composed of nodes (ActivityNodes) and edges (ActivityEdge). The metamodel extends the latter two allowing users to have collaborative activities such as nodes (CollaborativeActivityNode) and edges

(CollaborativeProcessEdge) that are activated from the operations that roles carry out. It can also be seen that collaborative nodes are related to a set of events that can trigger awareness information to be displayed in some collaborative element (Workspace, Tool, etc.).

Figure 3 displays the metaclasses to model protocols. The possibility of defining the behavior of a collaborative activity through a protocol is contemplated. The StateMachine metaclass is used to represent the protocol of an activity. The modeling consists in describing the states through which an activity passes, having into account the operations that roles can perform.

In the protocol metamodel, the metaclass CollaborativeActivityState, subclass of UML State, has a set of operations assigned through assignedRoleElementOperation and another set of operations that trigger a transition to another state. It is important to note that operations assigned to roles are related to an event that can cause an Awareness information update to be displayed in some collaborative element.

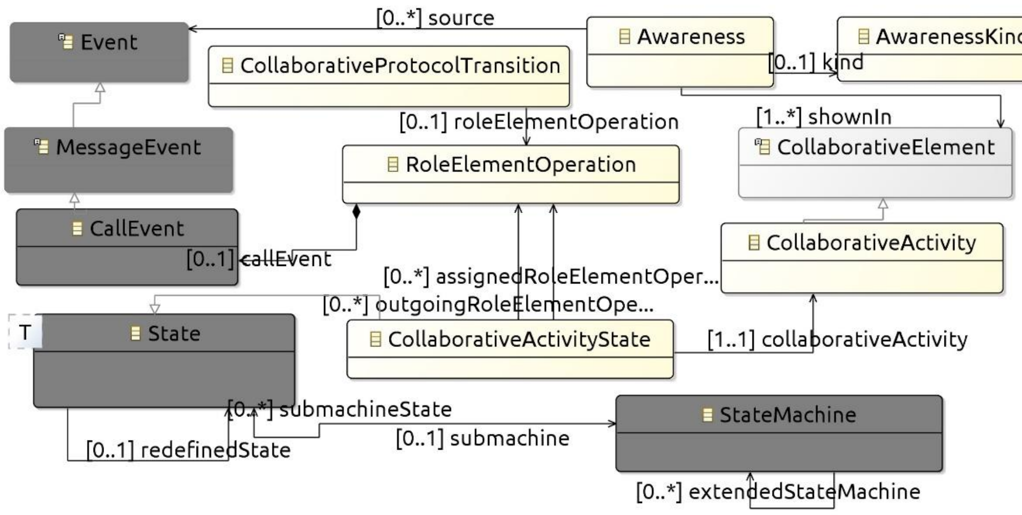


Figure 3: Protocols in CSSL

Concrete Syntax

The CSSL language was built extending UML, therefore it has a well-defined syntax. Using the metamodel, users can instantiate the CollaborativeModel metaclass that is a subclass of the UML Model, which contains the elements constituting the system as packaged elements (instances of PackageElement). This allows users to create CollaborativeRoles, CollaborativeActivity, Workspace, and so on.

Creating models from instantiating metaclasses, using a standard UML editor, is a hard and unfriendly work. This style of instantiation would require users to create an instance of CollaborativeModel and from it instantiate the "children" in that model, as shown in **Error! Reference source not found.** where, for example, Workspace is instantiated. In this case, abstract syntax is used as if it were concrete and a tree-shaped listing with all the instantiated classes and associations is displayed.

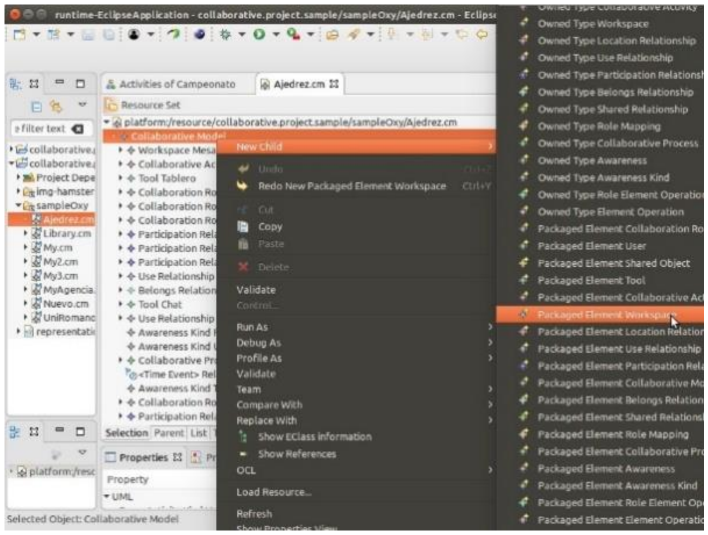


Figure 4: UML Editors

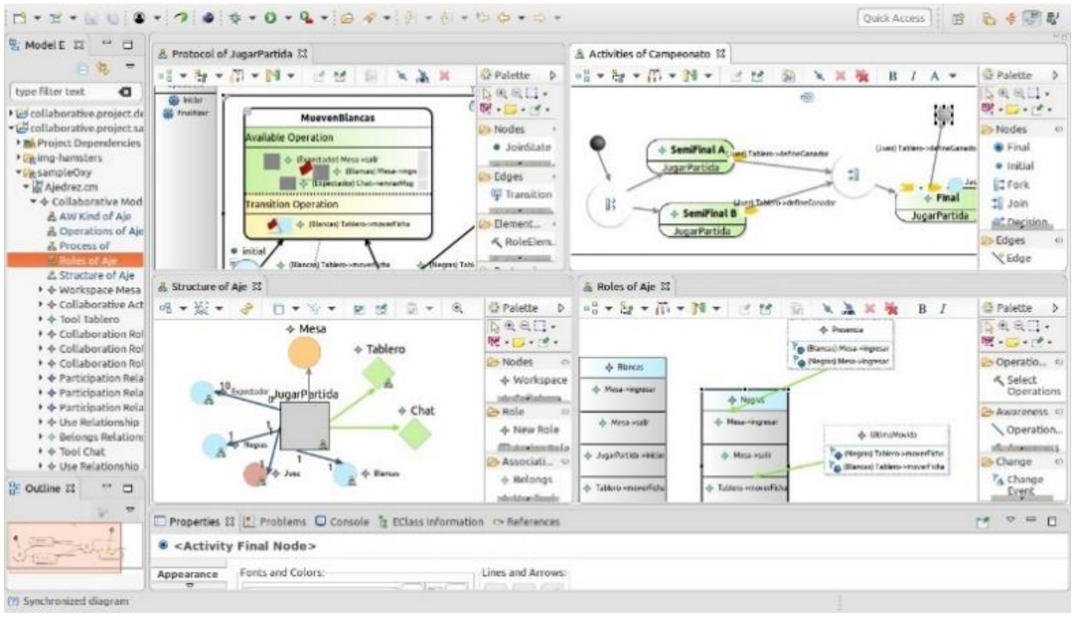


Figure 5: CSSL Language-Specific Editors

To provide more flexibility and readability to the DSL, different concrete syntaxes (Editors) can be offered. Then users can choose the syntax that is most intuitive for them. This work introduces a concrete syntax, implemented through graphical editors, that was developed using eclipse's Sirius project. It is shown in **Error! Reference source not found.**.

Different representations are available according to the needs of the language. The main graphical editors describing the collaborative system with awareness are:

- **System Structure:** this editor allows the user to create and connect the main concepts of the system. Awareness information is also supported.

- **System roles:** this editor allows the user to define which roles are involved in the system and which operations are assigned to them. The actions triggering the awareness are also defined.
- **Process Diagram:** For each process, the collaborative activities that make up it and in what order they are executed are displayed. In this editor, users can indicate how the awareness is updated as the process advances.
- **Activity diagram:** This editor allows the user to specify the states through which a collaborative activity goes. Users can also indicate how awareness information is modified due to the actions of the roles.

Other editors allow users to create processes, awareness types, add operations to spaces, add operations to activities, and other system configuration. Together they provide tools for designers to describe collaborative systems with awareness.

SEMANTICS OF THE DOMAIN-SPECIFIC LANGUAGE

MDA Initiative: PIM to PSM transformations

Model Driven Architecture (MDA) is an approach to software design, development and implementation guided by the Object Management Group (OMG) (Kennedy and Carter 2003)(Kennedy and Carter 2003; Kleppe, Warmer, and Bast 2003). MDA is a pragmatic realization of MDD and provides guidelines for structuring software specifications that are expressed as models. MDA separates business and application logic from underlying platform technology. Platform-independent models (PIM) of an application, built using UML or the other associated OMG modeling standards, can be realized through the MDA on virtually any platform. These platform-independent models document the business functionality and behavior of an application separated from the technology-specific code that implements, which is documented by Platform Specific Models (PSMs). Each one can evolve at its own rate: business logic responding to business need, and technology taking advantage of new developments. Therefore, interoperability both within and across platform boundaries is facilitated (Lidia Fuentes and Antonio Vallecillo 2004).

The CSSL language is independent of any deployment platform because it does not reference any of the features of specific technologies, such as: programming languages, hardware, network topology, and so on. Therefore, CSSL is a language for expressing PIMs.

The key challenge of MDA is in transforming PIMs to PSMs that can be used to generate implementations. Transforming a model into another model means that a source model is transformed into a target model based on some transformation rules, as shown in Figure 6.

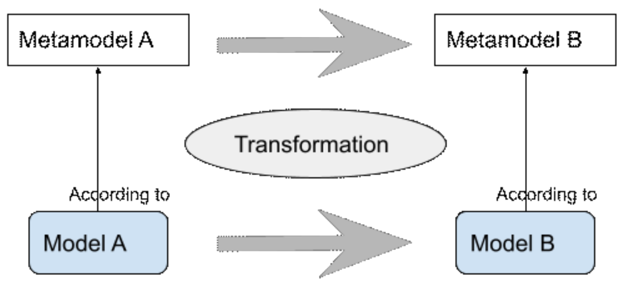


Figure 6: Transformation Architecture

One or more PSMs can be derived from a PIM, depending on the different technologies. In this section we describe a particular transformation from CSSL PIM models to an executable model. The transformation is composed by a set of rules associating each element of the CSSL Metamodel to an element in the target metamodel. In this case, the specific stack of technologies included in the PSM are JavaScript and technologies associated with that language, such as Node.js and client-side frameworks such as Angular and React. The main concepts conforming the PSM are Application, Component, Button, Process, Classes (Types), Permissions, Role, User and Container Component. Table 1 represents how elements of the PIM are transformed to the PSM.

After applying the transformation, the abstraction level is reduced because elements get defined in terms of elements in the chosen specific platform. Notice that not all elements from the PIM are mapped in Table 1.

Table 1: PIM to PSM transformation

Metaclases	Transformaction
CollaborativeModel	Application: A collaborative system or application is created for each model.
CollaborativeElement	Abstract Component: Collaborative elements will be visible components in the system. These elements have operations that appear as buttons in the system interface.
CollaborativeElement: Workspace	Container Component: A component visible on the system that will contain the activities that belong to it and the operations defined in its context.
CollaborativeElement: CollaborativeActivity	Container Component: A component visible on the system that will contain the tools that are used and the operations defined for the activity.
CollaborativeElement: Tool	Container Component: A component visible on the system that will contain the features defined for the tool.
CollaborativeElement: SharedObject	Container Component: A component visible on the system that will contain the operations defined for it.
ElementOperation	Button: the button appears on collaborative items and represents related operations that can be executed on each of them.
CollaborativeAssociation	Maintains relationships between collaborative elements.
CollaborativeAssociation: BelongRelationship	This association indicates that a CollaborativeActivity belongs to a Workspace.
CollaborativeAssociation: UseRelationship	This association indicates that a Tool is used in a CollaborativeActivity.
CollaborativeAssociation: ParticipationRelationship	This association indicates that a CollaborationRole participates in a CollaborativeActivity. This relationship defines which and how each role participates in each activity.
User	User registered in the system.
CollaborationRole	Role within the system. This element controls user permissions.
RoleElementOperation	Enables a role to execute an operation on a collaborative item. Makes the button visible to a particular role. On the other hand, it activates an Event indicating that the operation was executed. This is used to display awareness information somewhere in the system.
CollaborativeProcess	A container component composed of CollaborativesNodes instances, control nodes (initial, fork, join, decision, final), and edges connecting these nodes. It represents a system process.
CollaborativeActivityNode	Represents an instance of an activity within a process.
CollaborativeProcessEdge	The edges of a process that starts when a role executes an operation (RoleElementOperation).
CollaborativeActivityState	Represents a state within a collaborative activity. There is a set of RoleElementOperation assigned to it.
CollaborativeProtocol-Transition	It is a state change within an activity that originates from an operation (RoleElementOperation) performed in the source state.
AwarenessKind	Types of awareness information.

Awareness	A system-visible component that is linked to a component through the shownIn relationship. It defines a set of events that will update the component information.

Model-to-Text Transformation

The final step of the MDA is automatic code generation. By taking information from the PSM, an automatic transformation produces both structural and behavioral code that are compliant with the compilation process of the platform and can be executed on it. A target architecture was chosen to implement the concepts conforming the PSM.

This section introduces a Model-to-Text transformation developed with the Acceleo Eclipse plug-in (https://www.eclipse.org/acceleo/ n.d.). The transformation maps elements from the CSSL language to elements in the PSM model. The resulting PSM consists of a set of files containing code in some programming language and a target architecture; in this case a Web Client-Server architecture was chosen.

Specifically, for each element of the language, the transformation defines which system components must be built; both on the client and on the server side. On the other hand, a lightweight language was chosen that does not require complicated configuration (for example, JavaScript and technologies associated with that language, such as Node.js and client-side frameworks such as Angular or React).

The transformation also defines that the data exchange between the client and the server is performed through a REST API, which functions as an interface between systems that use the HTTP protocol to get the data or warn of the execution of some operation, using an XML or JSON format.

The target code is written in TypeScript, which is a JavaScript-based programming language, with the advantage of being a typed language, which allows the creation of class structures and can run on both the client and server side.

In short, the transformation consists of taking an instance of the CSSL language and producing as a result a Web application implemented in TypeScript, where the developers can then add their own code to adapt the result to their interests.

WebSocket technology was chosen to implement awareness. It establishes a bidirectional connection between the client's browser and the server that lasts as long as the client browser remains open. Both the client and server are able to send and receive messages over the established connection channel using the standard protocol WS or the secure version WSS.

Finally, executable models are obtained, through model transformations, using Javascript-related web technology (Express.js, Angular.js, Node.js), MongoDB and Websockets.

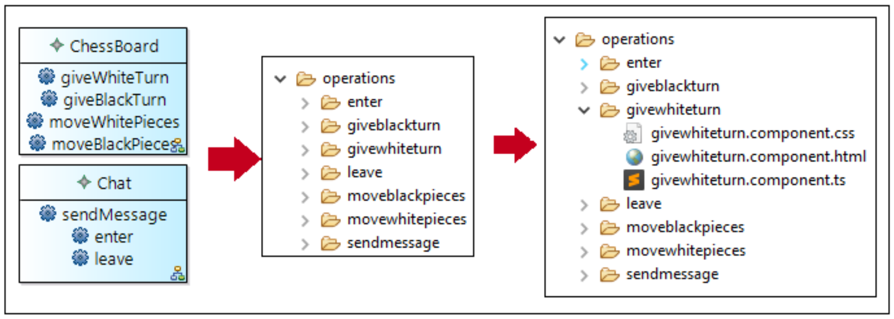
The main concerns of collaborative systems are already built in this preliminary version of the system. That is, user management, roles and operations, collaborative processes, and awareness are already implemented. Then, developers can refine this system according to their own needs.

Transformation output

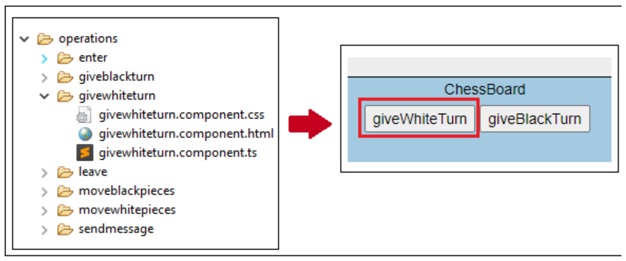
The target of the transformation is a web application that provides a set of basic functions that developers can refine to complete the development of the collaborative system. In this section the output of the transformation is analyzed in two steps. First the construction of the system structure is described and afterward the implementation of the system dynamics is explained.

System structure

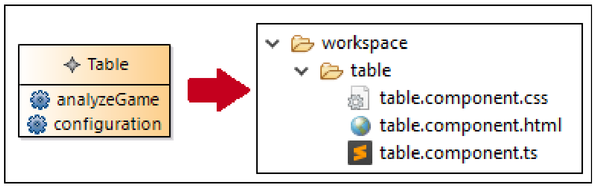
The main phases of transforming the structural aspects of the system are presented in this section:



(a)



(b)



(c)

Figure 7: (a) Strutre of Operations Transformations

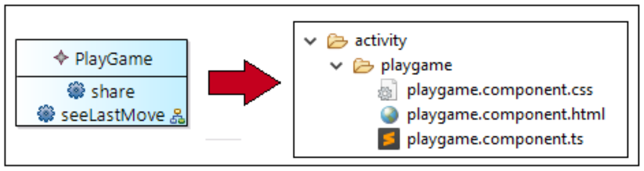
Figure 8: (b) Operation Transformation

Figure 9: (c) Workspace Transformation

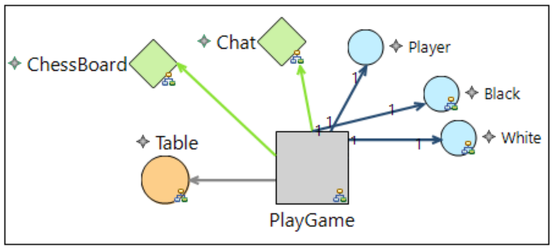
A directory default structure, based on the standards of each technology is obtained as it is shown in Figure 7.

- Transformation of operations. The operations are transformed into buttons in the application. The Figure 8 shows an example of a target application, a Chess game, that is obtained from two collaborative elements by applying the transformation.
- Workspace Transformation. Workspaces are transformed into containers that give a framework to the execution of activities. Additionally, containers hold their own operations. After the transformation is applied, Angular components are created, and saved in their proper directory ("workspace"). Figure 9 shows an example.
- Transformation of collaborative activities (CollaborativeActivity). Activities are located within workspaces. They are invocated by users within a collaborative process. Activities contain operations and offer tools to the users. Activities are transformed to Angular components that are saved in the "activity" directory within the project as shown in Figure 10.
- The belongRelationship, UseRelationShip, and ParticipationRelationship associations are used to connect the elements, getting the result shown in the Figure 11. The "PlayGame" activity appears in gray, the tools in green, the roles in light-blue color and finally the workspace is shown in

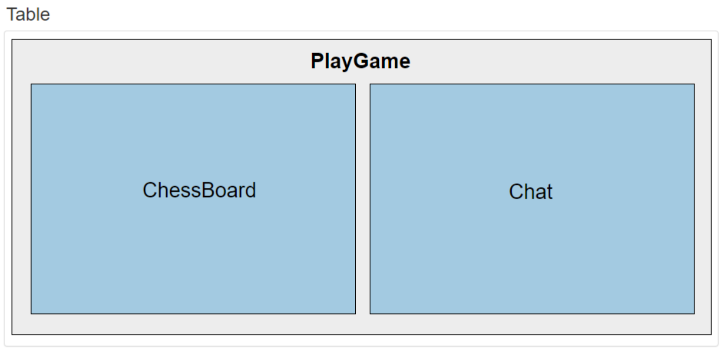
orange. The arrows show which elements are linked to each activity. These associations are instances of different associations expressed in the metamodel. The ParticipationRelationship relates the activity to the collaboration roles. The UseRelationship relates the activity to the tools. Finally, the BelongsRelationship relates the activity to the workspaces. All these relationships are processed by the transformation, given rise to the components and containers of the resulting collaborative system in the Figure 12.



(a)



(b)



(c)

Figure 10: (a) Activity Transformation

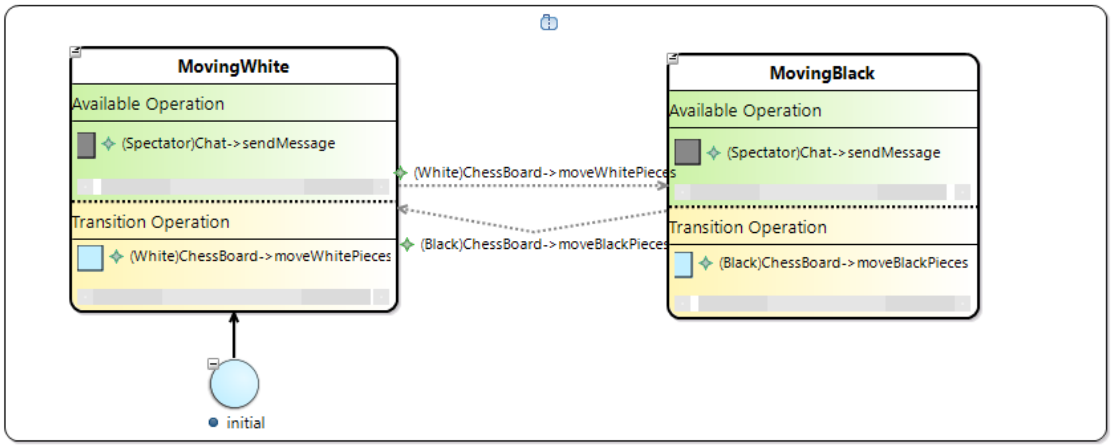
Figure 11: (b) Activity Relationships

Figure 12: (c) Activity Interface

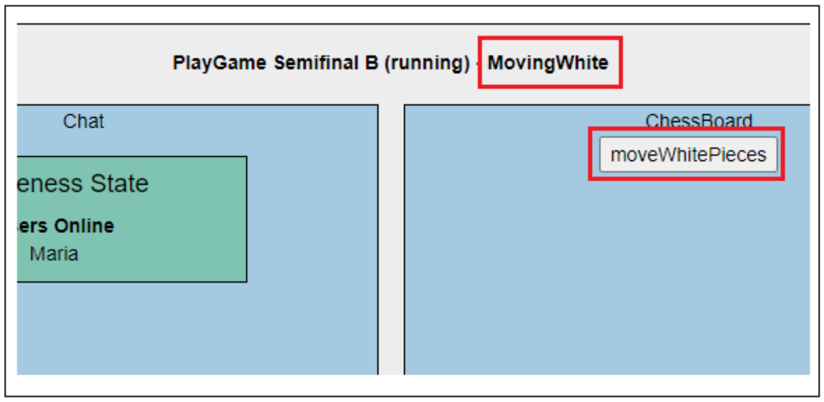
System dynamics

After finishing the first step of the transformation, a preliminary version of the web system is already available. It allows a group of users to connect to the system to share collaborative activities.

The next step of the transformation consists in implementing the dynamic aspects of the system. This section discusses the transformation of the main dynamic models: protocol construction, collaborative processes, and awareness.


MovingWhite
Available Operation
(Spectator)Chat->sendMessage
Transition Operation
(White)ChessBoard->moveWhitePieces
MovingBlack
Available Operation
(Spectator)Chat->sendMessage
Transition Operation
(Black)ChessBoard->moveBlackPieces
(White)ChessBoard->moveWhitePieces
(Black)ChessBoard->moveBlackPieces
initial

(a)



(b)

Figure 13: (a) Protocol Model

Figure 14: (b) Protocol Transformation

- **Transformation of activities protocols.** Protocols are modeled as state machines, and each state is specified as the set of operations that roles can execute in that state. Some of the operations can cause a state transition. In the example being followed, the "PlayGame" activity goes through the "Moving White" and "MovingBlacks" states. In the first, the Whites role can move a piece, and in the second the Black role is the one that can move, and the states change as the roles run the MovePiece operation, as shown in the Figure 13. In the interface, the user will always see the protocol status to the right of the activity name while the available operations appear as buttons that can be activated as shown in Figure 14.
- **Transformation of Collaborative Processes.** Processes organize a set of collaborative activities by defining in what order each of them is activated. When users enter the system, they will only have access to the processes in which they are enrolled. For example, Figure 15 shows the process that simulates a chess competition, where participants must play three games to win. In this process, three activities of the type "PlayGame" are created. As users enter the system, they are placed in the games they have to play, and the process starts. The system controls which user participates in each game and enables the winners to play the contest final.

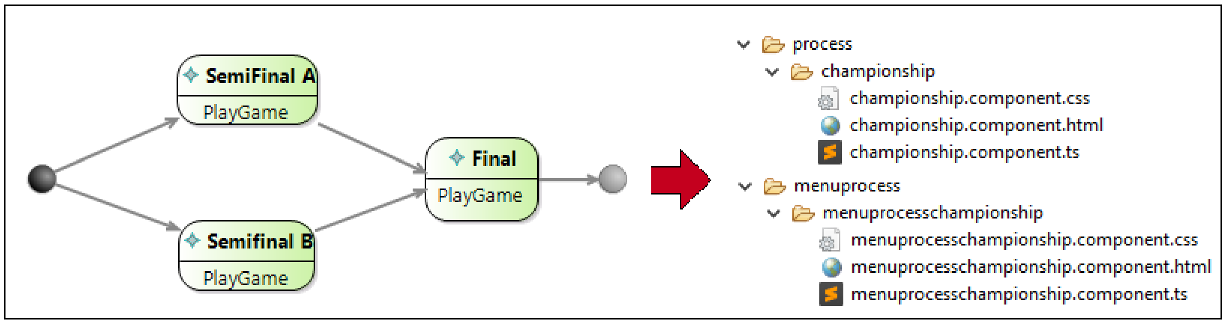
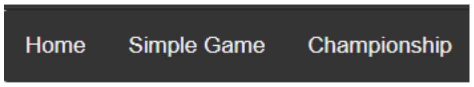
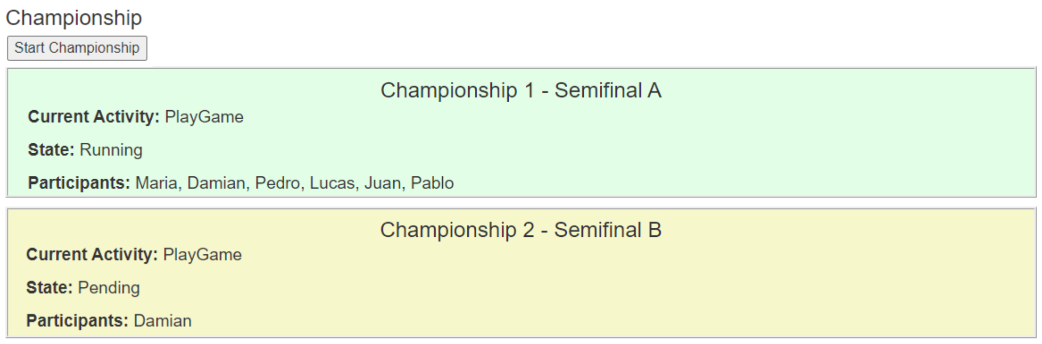

SemiFinal A
PlayGame
Semifinal B
PlayGame
Final
PlayGame
process
championship
championship.component.css
championship.component.html
championship.component.ts
menuprocess
menuprocesschampionship
menuprocesschampionship.component.css
menuprocesschampionship.component.html
menuprocesschampionship.component.ts

Figure 15: Collavorative Process Transformation

- **Creation of activities and processes.** One of the central decisions to make in the process of transforming from model to text is to define how system instances are to be created. In our example, the design decision was that users can create the activities and processes in which they participate. Once users are logged into the system, they can create Games and Championships from the system home. The example shows the user's games and championships, Figure 16. The system displays and supervises the status of each game as shown in Figure 17.



(a)



Championship 1 - Semifinal A

Current Activity: PlayGame

State: Running

Participants: Maria, Damian, Pedro, Lucas, Juan, Pablo

Championship 2 - Semifinal B

Current Activity: PlayGame

State: Pending

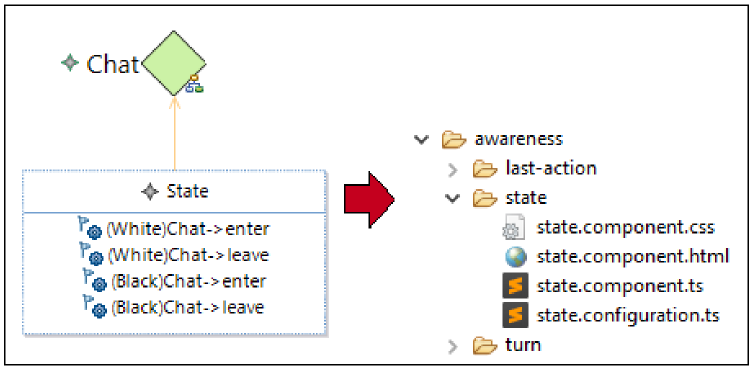
Participants: Damian

(b)

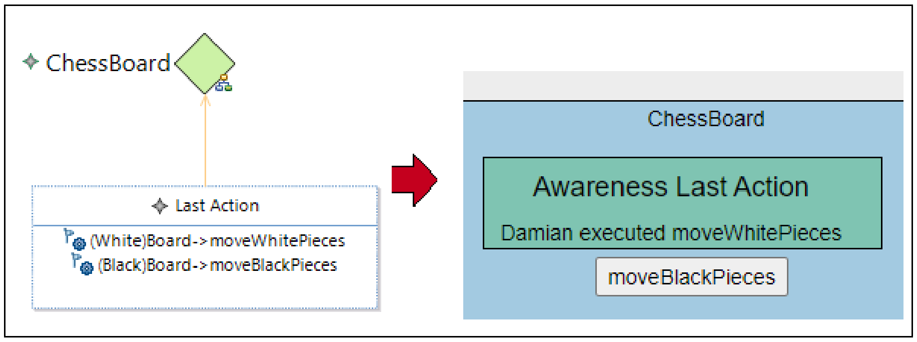
Figure 16: (a) Main Menu

Figure 17: (b) Process Instances

- **Transformation of awareness.** The transformation mechanism defines how those models are transformed into components that display the awareness information in the system, Figure 18. Awareness models will be transformed into components that will be displayed in collaborative elements (Workspace, CollaborativeActivity, Tool). At the same time, a communication channel (Websockets) is created between the awareness originator and the component that displays the information. The second example shows that the awareness "Turn" is displayed in the "Board" element and is updated from the user/role moves and the system clock, Figure 19.



(a)


ChessBoard
Last Action
(White)Board->moveWhitePieces
(Black)Board->moveBlackPieces
ChessBoard
Awareness Last Action
Damian executed moveWhitePieces
moveBlackPieces

(b)

Figure 18: (a) Awareness Transformation

Figure 19: (b) Last Action Awareness

CASE STUDY: Modeling the DimSum Thinklet

ThinkLets (Robert O. Briggs, Gert-Jan de Vreede, Gwendolyn L. Kolfschoten - 2007) are design patterns for collaborative work practices. ThinkLets are used by facilitators and collaboration engineers to design collaboration processes for high-value recurring task, and transferring those designs to practitioners to execute for themselves without the ongoing intervention of professional facilitators. The collection of thinkLets forms a pattern language for creating, documenting, communicating, and learning group process designs.

In (Robert O. Briggs, Gert-Jan de Vreede, Gwendolyn L. Kolfschoten - 2007) a fully documented thinkLet, named DimSum is presented as example. In this thinkLet, the team works to create a specific, precise, statement, in a way that all understand, and that accommodates the interests of all team members. Each member proposes a candidate terminology for the common statement. Participants then get the words and phrases they like best from the candidate statements to create a new common statement. Periodically all participants propose new candidate statements based on the current draft of the common statement. The cycle continues until a version emerges that all participants accept.

Here, the pattern of collaboration is a cycle of generation of concepts, clarification of meaning and consensus building on the final statement.

This collaboration pattern can be easily and accurately modeled by applying CSSL.

Before starting with the design of the interaction among the participants, the environment where the users collaborate should be described. The tools, the roles and the virtual place where the users will meet to develop the collaborative activities involved in the DimSum thinkLet are identified in a first step. Figure 20 displays the complete thinkLet environment. The model specifies two collaborative activities: one for generation of concepts (ContributeSt) and another for clarification of meaning and consensus building (ReviewSt). Both collaborative activities are carried out in the same virtual place called Room. Coordinator and participant are the two roles involved in the task. Finally, the specification describes the set of collaborative tools that users/roles employ to carry out the task. The tool set includes a. statement browser, a statement editor, a voice conference system, and a voting system.

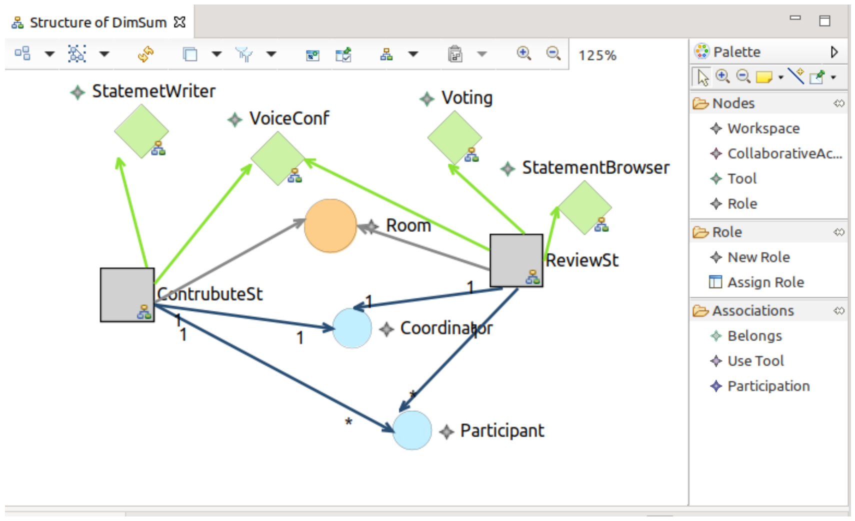


Figure 20: Modeling of the DimSum ThinkLet environment

The collaborative process model for the Dimsum Thinklet task is illustrated in Figure 21. The model specifies the ContributeSt and ReviewSt activities and the transitions that define the collaborative process. The activities are labeled with icons of the work of (Solano et al. 2014) that show the type of activity and whether they produce information or not.

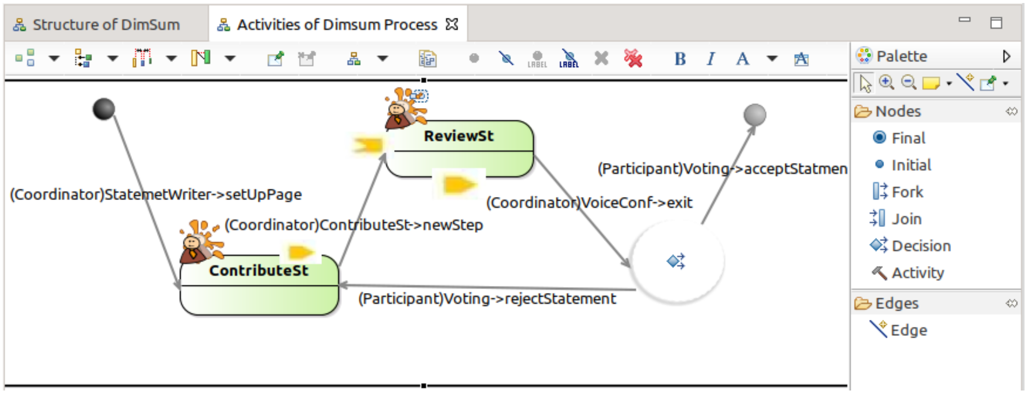

Structure of DimSum
Activities of Dimsum Process
ReviewSt
(Participant)Voting->acceptStatmen
(Coordinator)StatemetWriter->setUpPage
(Coordinator)VoiceConf->exit
(Coordinator)ContributeSt->newStep
ContributeSt
(Participant)Voting->rejectStatement
Palette
Nodes
Final
Initial
Fork
Join
Decision
Activity
Edges
Edge

Figure 21: Modeling of the DimSum ThinkLet process

Then, each of the collaborative activities is modeled in CSSL. For this purpose, the protocols are specified. Figure 22 shows the interactions between the participants in the ContributeSt collaborative activitiy. The model is compliant with the DimSum Thinklet specifications where it is described that the activity goes through different states. In each of them, users can perform different operations. The model shows a first state of the activity called Setup where the coordinator explains the objective of the activity. Then, in the following state, called ProduceSt, the participants contributions are generated and collected and finally in the last state, called ReviewSt, the candidate sentences are adjusted. The activity continues cyclically until a consensus is reached among all participants.

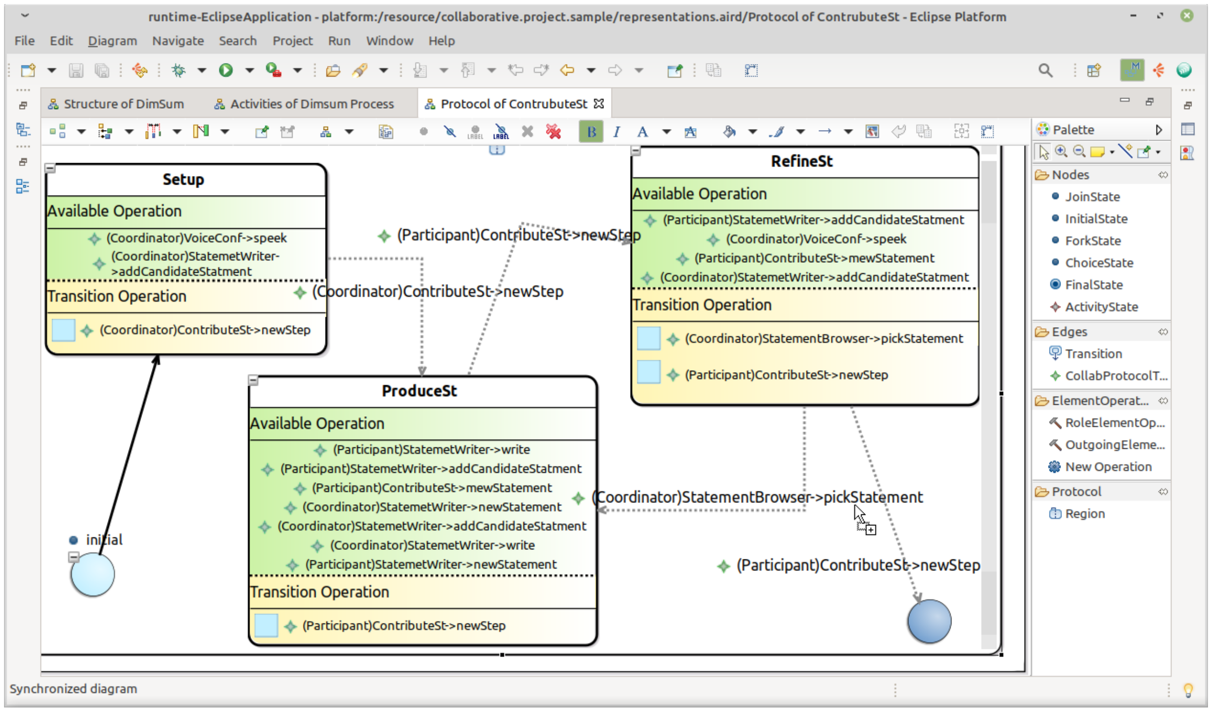

runtime-EclipseApplication - platform:/resource/collaborative.project.sample/representations.aird/Protocol of ContrubuteSt - Eclipse Platform
File Edit Diagram Navigate Search Project Run Window Help
Structure of DimSum
Activities of Dimsum Process
Protocol of ContrubuteSt
Setup
Available Operation
(Coordinator)VoiceConf->speek
(Coordinator)StatemetWriter->addCandidateStatment
Transition Operation
(Coordinator)ContributeSt->newStep
(Participant)ContributeSt->newStep
(Coordinator)ContributeSt->newStep
RefineSt
Available Operation
(Participant)StatemetWriter->addCandidateStatment
(Coordinator)VoiceConf->speek
(Participant)ContributeSt->newStatement
(Coordinator)StatemetWriter->addCandidateStatment
Transition Operation
(Coordinator)StatementBrowser->pickStatement
(Participant)ContributeSt->newStep
ProduceSt
Available Operation
(Participant)StatemetWriter->write
(Participant)StatemetWriter->addCandidateStatment
(Participant)ContributeSt->newStatement
(Coordinator)StatemetWriter->newStatement
(Coordinator)StatemetWriter->addCandidateStatment
(Coordinator)StatemetWriter->write
(Participant)StatemetWriter->newStatement
Transition Operation
(Participant)ContributeSt->newStep
(Coordinator)StatementBrowser->pickStatement
(Participant)ContributeSt->newStep
initial
Palette
Nodes
JoinState
InitialState
ForkState
ChoiceState
FinalState
ActivityState
Edges
Transition
CollabProtocolT...
ElementOperat...
RoleElementOp...
OutgoingEleme...
New Operation
Protocol
Region
Synchronized diagram

Figure 22: Modeling of the DimSum ThinkLet activity

Awareness Modeling

The above models describe the structure and dynamics of the DimSum thinkLet. But another important element in a collaborative process is the awareness specification. And indeed, the modeling of awareness is one of the most important strengths of CSSL since it allows the collaboration engineer to specify where the awareness components should be displayed, and which events trigger the update of each of them.

Although the awareness specification would notably enrich the thinkLet scope, the DimSum thinkLet does not define the types of awareness to apply. Therefore, the following five types of awareness are incorporated to the thinkLet model: Presence, Speaking, Statement, VotingResult, TimeToVote. Figure 23 shows the instances of the awareness types with the details of the events that update them.

It is noteworthy that most of the awareness were defined as synchronous and not transient (not stored in the system). Except in the case of the awareness Statements that must be stored in the system for later retrieval. The figure xx also shows that awareness components are linked to different types of elements (Workspace Room, VoiceConf Tool, Vooting Tool, ContributeSt Activity).

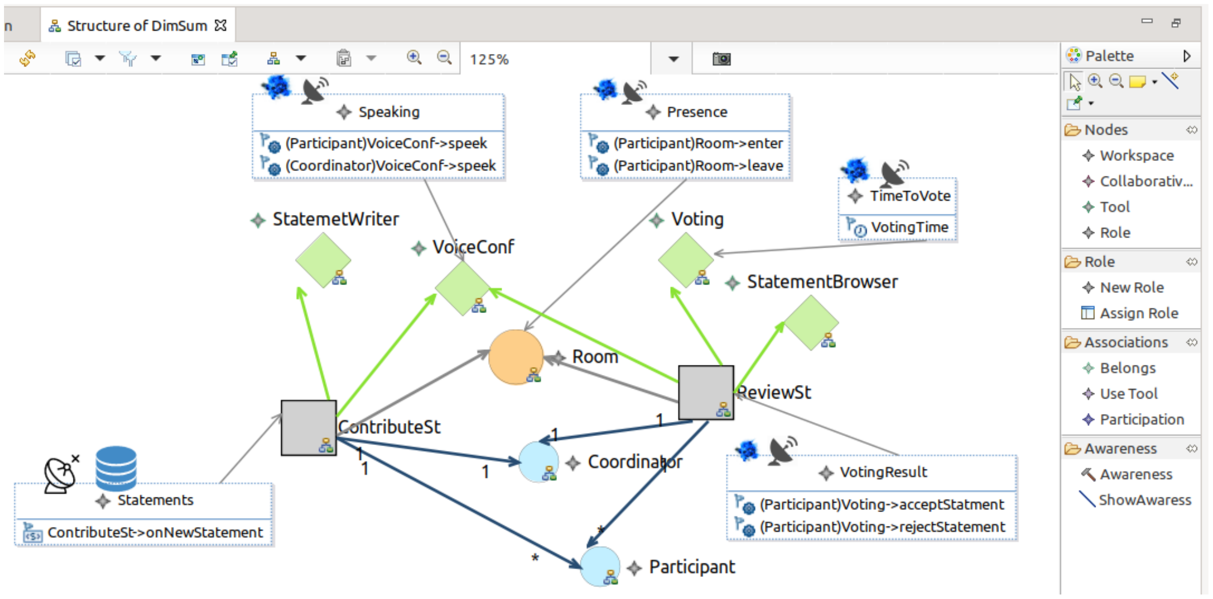


Figure 23: Modeling of the DimSum ThinkLet with Awareness

Most awareness updates are based on the actions that users perform. The system is attentive to the operations performed by the roles in the system and activates the update of awareness when appropriate. On the other hand, other awareness types are linked to events not originated by users. For example, the elapsed time, as specified in the Awareness TimeToVote, where there is a UML timeEvent activated to control the amount of time allowed to vote.

Awareness can also be specified linked to events in activities as shown in Figure 24.

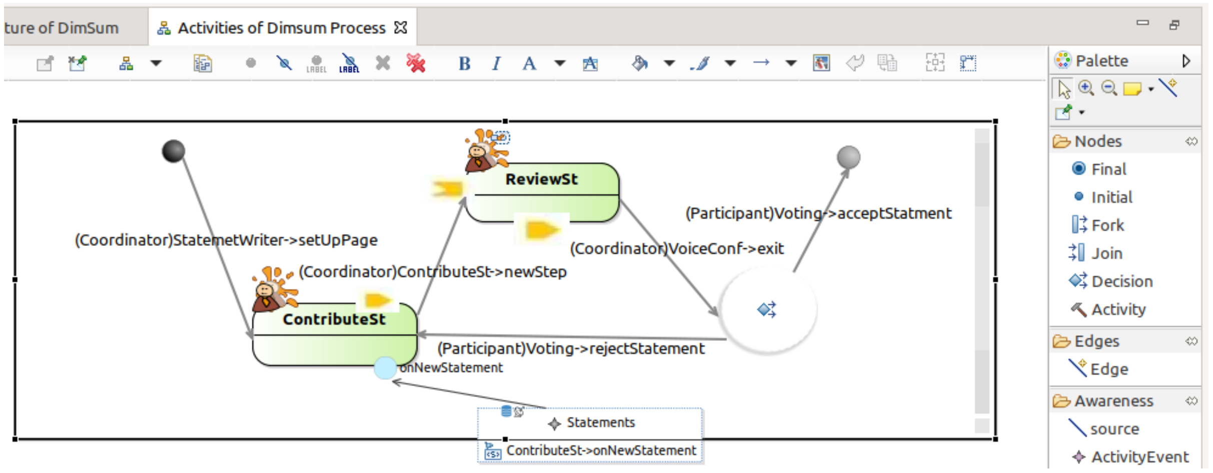

ture of DimSum
Activities of Dimsum Process
Palette
Nodes
Final
Initial
Fork
Join
Decision
Activity
Edges
Edge
Awareness
source
ActivityEvent
ReviewSt
(Participant)Voting->acceptStatment
(Coordinator)StatemetWriter->setUpPage
(Coordinator)VoiceConf->exit
(Coordinator)ContributeSt->newStep
ContributeSt
(Participant)Voting->rejectStatement
onNewStatement
Statements
ContributeSt->onNewStatement

Figure 24: Modeling of the DimSum ThinkLet process with awareness

COMPARISION WTH RELATED WORKS

CSSL is a domain specific language built as a UML extension using the metamodeling mechanism. A first version of the language was introduced in (Bibbo, Giandini, and Pons 2017). The current version meets all the requirements stated in the related works section.

Unlike de work of (Gallardo et al. 2012) with CSSL, the content of the editors cannot be specifically modeled, but CSSL allows integrating any type of collaborative tool into the design, describing what operations the roles can execute at each time. Regarding the workspace, CSSL allows engineers to design relationships between workspaces such as the inclusion between workspaces. This facility allows covering several collaborative domains (E-learning, Brainstorming, Collaborative games, etc.). It also allows integrating a specific design of the awareness information associated with the different elements of the model.

Although the proposal of using thinklets (Briggs, Gert-Jan de Vreede, and Kolfschoten 2007) is interesting to document the dynamics of collaboration, CSSL provides a more comprehensive view of the system. In addition to modeling the interaction, CSSL defines the collaborative tools (Tools) that will be used at each time, the collaboration environment (Workspace) and especially the awareness modeling, indicating the type of information and in which collaborative element it will be displayed and how it will be kept up to date. An example of a thinklet modeling with CSSL is presented in this article, as a case study.

The difference between CSSL and most related proposals is that CSSL is formally linked to UML. CSSL extends some UML classes that allow modeling collaborative processes (extending Activity) and protocols (extending StateMachine). These designs combine the power of the UML to describe processes and protocols (with their metaclasses of Fork, Join, Decision, etc.) with the actions (operations) performed by the roles in collaborative activities.

The popularity and maturity of UML favors that CSSL can be widely adopted for modeling collaborative systems. Also, it brings the benefits of leveraging the available UML modeling software tools.

Regarding language coverage, the expressive power of CSSL is enough to represent all surveyed concepts. In addition, in CSSL the awareness modeling smoothly integrated into the rest of the models gives rise to a new and powerful design that allows obtaining a concrete implementation automatically through model transformations adhering to the model driven software development approach. And then getting benefits in terms of standardization, code reuse and automation.

EVALUATION, VALIDATION AND VERIFICATION OR THE METAMODEL

Evaluation of the Metamodel

This section performs a metamodel evaluation based on metrics recognized in the academic community. For this analysis, a set of three metrics presented in (Robert et al. 2009) was taken as the main reference. The metrics quantitatively determine how good a UML Profile is. Given their characteristics, these metrics can also be applied to metamodels, measuring a) How close the metamodels is to UML semantics, b) How complex it is, c) How configurable it is.

The metrics are:

- **ANLA** (Average Number Location Application): This metrics calculates the average number of UML elements to which stereotypes in a UML Profile can be applied. It is calculated as:

$$ANLA = 1/n \sum_{i=1}^{n} Mi$$

Where n is the number of stereotypes conforming the profile and Mi is the amount of metaclasses to which the i-th stereotype can be applied. The stereotype can be applied to any subclass of its base metaclasses. Therefore, a larger ANLA will be obtained when the affected metaclasses are closer to the "root" element of the metamodel referenced by the profile than when they are closer to the “leaves". For example, a stereotype that extends the Element metaclass can be applied to 199 metaclasses. The ANLA gives an indication of how much the profile depends on the UML metamodel. A high ANLA represents a profile that extends metaclasses close to the root elements of the metamodel, and because the semantics of them are more general, the resulting profile will not be as dependent on the UML metamodel. In contrast, a low ANLA represents a more dependent profile.

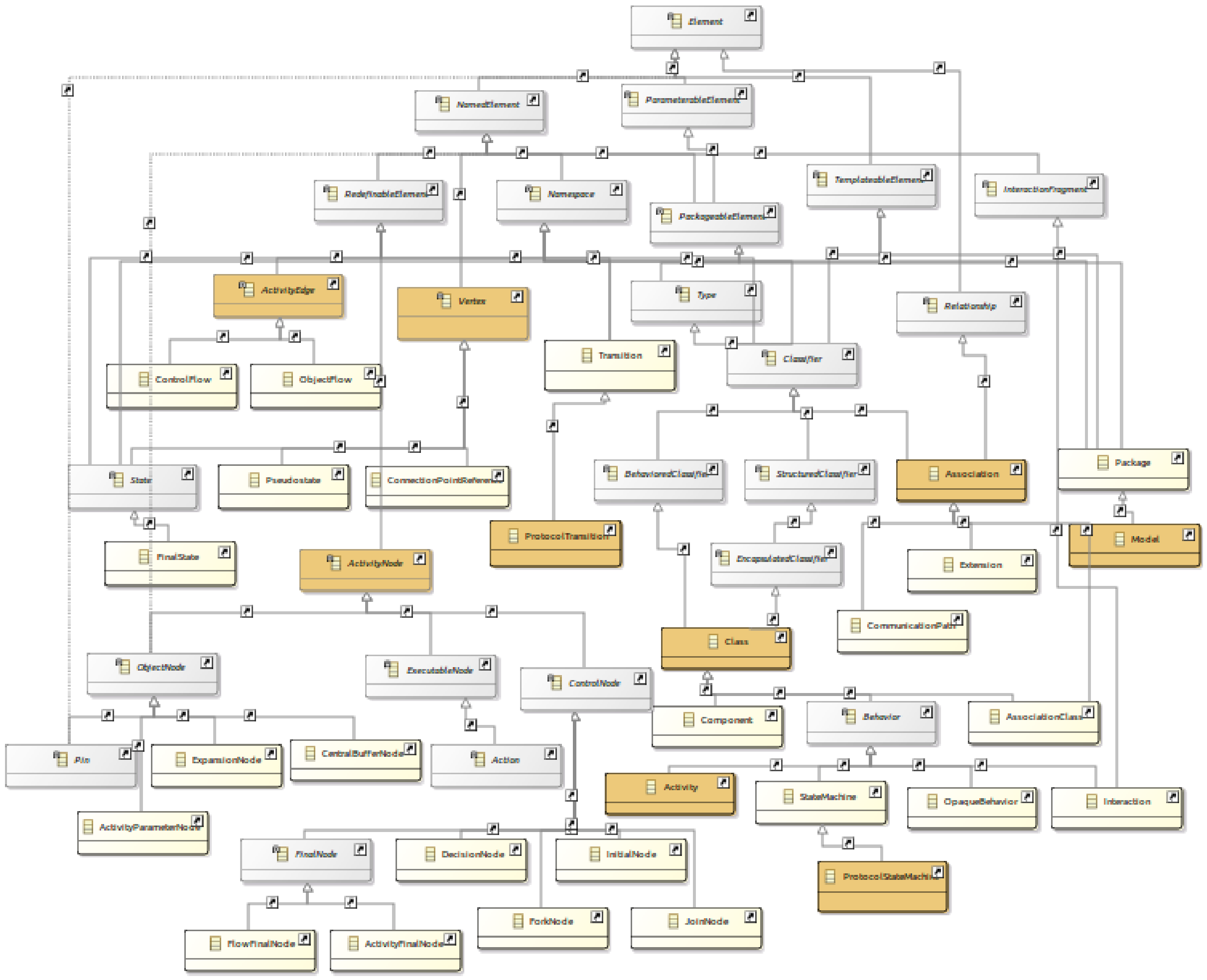


Figure 25: UML metaclasses to calculate ANLA

For the metric to be applicable to a metamodel, a variant of the formula was defined. According to the adapted metric each metaclass will be analyzed to determine whether it extends a UML metaclass near the root or closer to the leaves. The distribution of extended metaclases in the hierarchy tree can be seen in the Figure 25, where the main UML metaclasses that are extended by the CSSL language are highlighted. Specifically, for each metaclass, the adapted metric calculates two distances: - RI that is the number of subclasses ranging from the extended metaclass to the farthest leaf and - RS that is the length of the path to the root. For example, for a metaclass that extends the Class metaclass, RI is 3, while RS is 7. The variant of the ANLA formula is as follws:

$$ANLA = \sum_{i=0}^{n} RIi/(RSi + RIi)$$

- ALDIT (Average Leaf Depth of Inheritance Tree): This metric represents the average length of the inheritance tree in stereotypes. To apply the ALDIT metric to the metamodel, the metaclasses that extend a CSSL metaclass are counted and the average is calculated using the following formula:

$$ALDIT = \sum_{i=0}^{n} Li$$

Where n is the number of metaclasses in the CSSL metamodel and for each of them, Li = 1 if the metaclass extends other CSSL metaclass or Li=0 if it extends a UML metaclass.

- ASWA (Average Stereotype with Attributes): This metric calculates the average of stereotypes that have attributes. To apply ASWA to the CSSL metamodel, CSSL metaclasses that have attributes are counted and the formula is defined as follows:

$$ASWA = \sum_{i=0}^{n} Ai$$

Where n is the number of CSSL metaclasses and Ai = 1 if the metaclass has one or more attributes, or Ai = 0 in other case.

Applying the metrics to CSSL

In this section the results of applying the metrics on the CSSL metamodel are shown. Considering that he CSSL metamodel defines 25 new metaclasses, the index **n is 25**.

- The formula ANLA {ANLA}, where each Mi is calculated as follows.
    - For the 12 metaclasses extending the metaclass "Class", the RI is 3 and the RS is 7. Then Mi = (3/(3+7)) = 0.3
    - For the metaclass that extends "Model" the RI is 0. Then Mi = 0.
    - For the 6 metaclasses that extend"Association" the RI is 1 and the RS is 2. Then Mi = (1/(1+2)) = 0.3333.
    - For the metaclass extending ActivityNode, the RI is 3 and the RS is 3, then Mi = 0.5.
    - For the metaclass that extends ActivityEdge, the RI is 1 and the RS is 3, then Mi = 0.25.
    - For the metaclass that extends Vertex, the RI is 2 and the RS is 2, then Mi = 0.5.
    - For the 3 metaclasses that extend ProtocolTransition, Activity, and ProtocolStateMachine respectively, the RI is 0. Then Mi = 0.

- In summary, **ANLA = 0.274**

This result is relatively low, indicating that the language mostly extends classes close to the leaves of UML metamodel. According to Robert et al.'s work (Robert et al. 2009), this means that the CSSL metamodel relies heavily on UML semantics, allowing the use of tools implemented for the management of UML models in the management of CSSL models, such as editors, code generators, profiles, etc.

- **ALDIT:** calculating this metric, Li = 1 for 9 CSSL metaclasses that extends other CSSL metaclasses, while Li = 0 for the remaining16 classes. Therefore:
    - **ALDIT = 9/25 = 0.36**

- **ASWA:** the amount of CSSL metaclasses that have some attribute is 14, then:
    - **ASWA = 14/25 = 0.56**

The application of ALDIT and ASWA, shows that the CSSL metamodel has a low complexity **(0.36)**, with a short inheritance tree. On the other hand, it provides good parameterization possibilities **(0.56)** by users since more than half of the metaclasses have useful attributes for customizing the models to be instantiated from CSSL.

Conceptual Model Validation

According to Sargent (Sargent 2013), conceptual model validity means determining that (1) theories and assumptions underlying the conceptual model are correct, and (2) model representation of the problem, structure, logic, and mathematical and causal relationships of the model are reasonable, for the intended purpose of the model.

Then Sargent in (Sargent 2013) claims that the theories and assumptions underlying the construction of the model are tested using mathematical analyses and statistical methods. And that the opinion of experts on the subject is essential to determine the validity of the conceptual model. For this reason a systematic literature review was carried out to corroborate that experts accept the concepts, structures, and relationships included in the CSSL metamodel.

Through the systematic literature review (Bibbo et al. 2016), different models and ontologies of Collaborative Systems were analyzed, identifying a set of representative conceptual elements., such as "Shared Object", "Tool", "Collaborative Activity","Workspace", "User/Role", "Group". These elements also relate to the concept of awareness. As described in (Belkadi et al. 2013), one of the questions used to identify awareness is: "What roles will the other members of the group assume?" where the association between the concept of role and awareness appears. In other examples of awareness the question to ask is: “How can I help other participants to complete the project?” or “What are they doing?” or “Where are they?” In addition, most works incorporate the concept of "Task", which represents the collaborative tasks that users must perform.

The literature review and the related works corroborates that the CSSL metamodel presented in this paper, contemplates and includes all the concepts presented in the different scientific works. A noteworthy aspect is that the CSSL metamodel contemplates the concept "Awareness" related to "Collaborative Elements", "Collaborative Process" and "Protocol", allowing the creation of designs that include "Awareness" in the models instantiated from the metamodel.

In conclusion, the CSSL conceptual model can be considered correct and reasonable for its purpose, and therefore valid. This claim is supported by the systematic review of the work of numerous experts in the field of collaborative systems with awareness.

Model Verification

Sargent explains in (Sargent 2013) that model verification can be performed through computerized verification. This ensures that the computer programming and implementation of the conceptual model are correct. This means that the models are implementable and that the programs that are developed from these models work. To help ensure that a correct software is obtained, software development techniques found in the field of software engineering should be used in developing and implementing the software.

According to this definition, the CSSL metamodel can be considered verified because the semantics of the metamodel was implemented by means of a model-to-text transformation, as described in section 4. The transformation produces an executable web version of the model that solves central aspects of collaborative systems, as follows:

- Registration of users/roles in the system.
- Security issues, session management, login, and logout.
- Allow users/roles to perform operations provided by the collaborative item.
- Allow users/roles to access collaborative spaces, participate in collaborative activities and use collaborative tools.
- Coordinate user participation within collaborative activities according to protocols.
- Schedule collaborative activities in collaborative processes.
- Manage in-system instantiating.
- Providing an Awareness Platform.
- Construction of a prototype of the Application.

Once the program has been developed and possible bugs have been fixed, the first tests can be performed to determine if the system is working correctly. At this stage, different types of tests ranging from unit test, integration test to user acceptance test are performed. For example, the intuitiveness of the navigation between collaborative spaces, the usefulness of the awareness, and the correctness of the collaborative activities are subjects that can be analyzed at this stage.

At this point the system requirements are validated and design alternatives are considered to achieve a properly functioning application.

When modeling errors are detected, the analysts are able to correct the CSSL models and generate the modified system automatically by re-applying the model-to-text transformation. This demonstrates the advantages of MDD by allowing iterating between modeling and testing several times until obtaining the system that meets the requirements.

CONCLUSION AND FUTURE WORK

Collaborative Systems are applications in which a group of users, following a common goal, develop different joint activities. On the one hand the collaborative software allows users to share information, communicate, and coordinate joint activities and, on the other hand, it provides awareness information, which requires maintaining and informing users about the status of the collaboration on the fly.

Building these systems is a very complex task that requires powerful tools.

In this paper a proposal to apply the Model Driven software Development approach (MDD) to the construction of Collaborative systems was elaborated.

MDD separates business and application logic from underlying platform technology. Models allow designers to specify the solution using problem domain concepts directly. End products are then automatically generated from these high-level specifications.

The proposal consists of a domain specific language, named CSSL v2.0, that defines the main concepts of collaborative systems, especially collaborative processes, protocols and awareness. A concrete syntax is defined via a set of editors through which collaborative systems models are created by instantiating the language concepts. In addition to the static aspects, dynamic aspects such as collaborative processes, protocols and awareness are modeled.

Following the MDD methodology, the language semantics was defined through model-to-text transformations, obtaining executable code from the models expressed in the CSSL v2.0 language.

The target of the transformation is a web application that provides a set of basic functions that developers can refine to complete the development of the collaborative system.

The transformation was implemented in Acceleo and for the target code a lightweight and flexible javascript-based technology was used (Express, Node.js, Angular, MongoDB and Websockets).

The main concepts of the collaborative system were mapped into application components, both on the server and on the client side. The model-to-text transformation allows developers to obtain a draft version of the collaborative system that resolves the management of users, roles and operations, collaborative processes and awareness.

The results obtained corroborate that the CSSL v2.0 language allows defining in a precise, concise, and friendly way the abstract concepts of collaborative systems, including collaborative processes, protocols and awareness. This allows designers to build models that facilitate communication between developers, and executable versions are derived from them. An important aspect to highlight is that the models are built maintaining compatibility with UML, which makes it possible to interchange with other tools, such as editors, translators, or other UML profiles. This allows designers to use tools developed for CSSL and UML in combination.

Finally, evaluation, validation and verification of the language were carried out. The evaluation determined that the CSSL metamodel has low complexity, with semantics strongly associated with UML and with good configuration possibilities. On the other hand, the validity of the metamodel was analyzed based on a systematic review of the work of different experts from the scientific community determining that concepts and relationships of the metamodel are reasonable for its intended purpose. Finally, model verification was performed through computerized verification, demonstrating that the model-to-text transformation allows developers to obtain an executable web system that solves central aspects of collaborative systems.

Future work includes the combined application of other UML profiles complementing the CSSL language, such as Mobile profiles. Additionally, other transformations will be built from the models to obtain implementations on different platforms.

REFERENCES

Belkadi, Farouk, Eric Bonjour, Mauricio Camargo, Nadège Troussier, and Benoit Eynard. 2013. “A Situation Model to Support Awareness in Collaborative Design.” *International Journal of Human Computer Studies*.

Bibbo, Luis Mariano, Roxana Giandini, and Claudia Pons. 2016. “Sistemas Colaborativos Con Awareness: Requisitos Para Su Modelado.” *XLV Jornadas Argentinas de Informática e Investigación Operativa (45 JAIIO) - Simposio Argentino de Ingeniería de Software (ASSE 2016)* (Ciencias Informáticas):111–22.

Briggs, Robert Owen, Gert-Jan de Vreede, and Gwendolyn Kolfschoten. 2007. “ThinkLets for E-Collaboration.” *Igi-Global.Com*.

Collazos, César A., Francisco L. Gutiérrez, Jesús Gallardo, Manuel Ortega, Habib M. Fardoun, and Ana Isabel Molina. 2019. “Descriptive Theory of Awareness for Groupware Development.” *Journal of Ambient Intelligence and Humanized Computing* 10(12):4789–4818.

Dourish, Paul and Victoria Bellotti. 1992. “Awareness and Coordination in Shared Workspaces.” *Proceedings of the 1992 ACM Conference on Computer-Supported Cooperative Work - CSCW ’92* 107–14.

Dourish, Paul and Sara Bly. 1992. “Portholes: Supporting Awareness in a Distributed Work Group.” *Dl.Acm.Org*.

Ellis, Clarence A., Simon J. Gibbs, and Gail Rein. 1991. “Groupware: Some Issues and Experiences.” *Communications of the ACM* 34(1):39–58.

Gallardo, Jesús, Crescencio Bravo, and Miguel A. Redondo. 2012. “A Model-Driven Development Method for Collaborative Modeling Tools.” *Journal of Network and Computer Applications* 35(3):1086–1105.

Gallardo, Jesús, Ana I. Molina, Crescencio Bravo, Miguel A. Redondo, and César A. Collazos. 2011. “An Ontological Conceptualization Approach for Awareness in Domain-Independent Collaborative Modeling Systems: Application to a Model-Driven Development Method.” in *Expert Systems with Applications*.

Gallego, Fernando, Ana Isabel Molina, Jesús Gallardo, and Crescencio Bravo. 2011. “A Conceptual Framework for Modeling Awareness Mechanisms in Collaborative Systems.” Pp. 454–57 in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 6949.

Grudin, Jonathan. 1994. “Computer Supported Cooperative Work: History and Focus.” *Computer*.

Grudin, Jonathan and Steven E. Poltrock. 1997. “Computer-Supported Cooperative Work and Groupware.” *Advances in Computers* 45:269–320.

Gutwin, Carl and Saul Greenberg. 2002. “A Descriptive Framework of Workspace Awareness for Real-Time Groupware.” *Computer Supported Cooperative Work*.

Gutwin, Carl, Saul Greenberg, and Mark Roseman. 1996. “Workspace Awareness in Real-Time Distributed Groupware: Framework, Widgets, and Evaluation.” in *Proceedings of HCI ’96*.

https://www.eclipse.org/acceleo/. n.d. “Acceleo | Home.” Retrieved April 30, 2019 (https://www.eclipse.org/acceleo/).

Kamoun, Aymen, Saïd Tazi, and Khalil Drira. 2012. “FADYRCOS, a Semantic Interoperability Framework for Collaborative Model-Based Dynamic Reconfiguration of Networked Services.” *Computers in Industry* 63(8):756–65.

Kennedy, Alan and Kennedy Carter. 2003. “MDA Guide Version 1.0.1.” *Object Management Group*.

Kleppe, Anneke, J. Warmer, and Wim. Bast. 2003. *The Model Driven Architecture: Practice and Promise. 2003*. Addison-Wesley.

Lidia Fuentes and Antonio Vallecillo. 2004. "An Introduction to UML Profiles." *Cepis.Org/Upgrade* (II (04)).

Molina, Ana I., Jesús Gallardo, Miguel A. Redondo, Manuel Ortega, and William J. Giraldo. 2013. "Metamodel-Driven Definition of a Visual Modeling Language for Specifying Interactive Groupware Applications: An Empirical Study." *Journal of Systems and Software* 86(7).

Robert, Sylvain, Sébastien Gérard, François Terrier, and François Lagarde. 2009. "A Lightweight Approach for Domain-Specific Modeling Languages Design." Pp. 155–61 in *2009 35th Euromicro Conference on Software Engineering and Advanced Applications*. IEEE.

Sargent, R. G. 2013. "Verification and Validation of Simulation Models." *Journal of Simulation* 7(1):12–24.

Solano, Andrés, Toni Granollers, César A. Collazos, and Cristian Rusu. 2014. "Proposing Formal Notation for Modeling Collaborative Processes Extending HAMSTERS Notation." Pp. 257–66 in *Advances in Intelligent Systems and Computing*. Vol. 275 AISC. Springer Verlag.

Stahl, Thomas, Markus Völter, Jorn Bettin, Arno Haase, and Simon Helsen. 2006. *Model-Driven Software Development - Technology, Engineering, Management*. Pitman.

Teruel, Miguel A., Elena Navarro, V'\ictor López-Jaquero, Francisco Montero, and Pascual González. 2011. "An Extension of i * to Model Requirements for CSCW Systems Applied to Conference Preparation System with Collaborative Reviews." Pp. 84–89 in *5th International i* Workshop (iStar'11)*.

Teruel, Miguel A., Elena Navarro, Víctor López-Jaquero, Francisco Montero, and Pascual González. 2013. "CSRML Tool: A Visual Studio Extension for Modeling CSCW Requirements." in *CEUR Workshop Proceedings*.

Teruel, Miguel A., Elena Navarro, Víctor López-Jaquero, Francisco Montero, and Pascual González. 2014. "A CSCW Requirements Engineering CASE Tool: Development and Usability Evaluation." *Information and Software Technology* 56(8).

Teruel, Miguel A., Elena Navarro, Víctor López-Jaquero, Francisco Montero, Javier Jaen, and Pascual González. 2012. "Analyzing the Understandability of Requirements Engineering Languages for CSCW Systems: A Family of Experiments." *Information and Software Technology*.

Vieira, Vaninha, Patricia Tedesco, and Ana Carolina Salgado. 2010. "Using a Metamodel to Design Structural and Behavioral Aspects in Context-Sensitive Groupware." Pp. 59–64 in *Proceedings of the 2010 14th International Conference on Computer Supported Cooperative Work in Design, CSCWD 2010*.