An. Şt. Univ. Ovidius Constanţa Vol. 11(1), 2003, 113–124

META-MODELLING BASED SPECIFICATIONS FOR ADAPTIVE COLLABORATIVE ENVIRONMENTS

Cristina Mândruţă

To Professor Silviu Sburlan, at his 60's anniversary

Abstract

The paper presents a metamodel for open distributed collaborative systems. Based on this metamodel there are also presented the specifications of a framework for collaborative applications dynamic development and execution and for adaptive integration of agents in collaborative applications.This framework allows both dynamic modelling of collaborative applications and dynamic modelling of support API in order to allow any application definitions. The framework contains four typical platform services as support for collaborative executions and a service based on metamodelling that offers possibilities for modelling and dynamic building of system entities. It contains a communication model for the system components that allows adaptive integration of agents based on a negotiation process. The model for data communication is that of a shared information space managed by a specific service of the framework. Modelling abilities offered by the framework are the basis for collaborative applications dynamic development and for adaptive integration of agents in the collaborative application. When integrating a new agent, the system functionality evolves by extending the system interface semantics. As the application develops, the agent may "learn" new functions, according to system needs. It also may "remember" what it already knew when entering the system. The framework has the possibility to develop a "socializing" mechanism. The agents must adapt their functionality to the agents society needs. This mechanism offers to the agents a set of facilities to negotiate the accepted functions and the collaboration rules so that we can tell that the agents integration process take into account their "personality" and they can also readapt in the future.

Key Words: Meta-modelling, collaborative environments, distributed agents systems, adaptive integration, negotiation.

1 Introduction

The existing collaborative frameworks offer different types of support for collaborative activity. The static ones are concerned with collaborative applications execution [8]. The more flexible ones allow dynamic application development and dynamic integration of new agents in the collaboration.

A Process Coordination Framework [1] presents a multilevel specifications model used in e-business activities that allows the integration of different Web services in collaboration. Distributed architectures for software modelling and execution [5] allow collaborative software dynamic development. Agent-based workflow architecture [6,7] is a support for a services workflow building and executing. This architecture is flexible in the direction of the workflow agent's adaptation by global efficiency and correctness features negotiation. Different distributed agents software systems allow extending the system interface semantics by integrating new agents [4,9].

The paper presents the specifications of a collaborative framework that has even higher flexibility because it allows both dynamic modelling of collaborative applications and dynamic modelling of support API in order to allow any application definitions.

A collaborative distributed system may be viewed as a society of collaborative agents where an agent is a general concept referring to a significant functional element of a distributed application. Another feature of the framework presented in this paper refers to a complex type of agent integration in the agent's society, based both on society needs and on agent " personality" , and realized by a negotiation mechanism. There are at least three integration models. The first model integrates new agents by creating them and providing them with a functionality defined by the society. The second model integrates new agents by extending the functionality of the society with the whole new agent functionality [1]. The third model, proposed in [10] integrates new agents by integrating a part of the new agent offered functionality. This was called adaptive integration and is realized in a negotiation process based on new agent offered features and agent's society needs.

2 Formal Model For Open Distributed Collaborative Systems

2.1 Interface model

From functionality point of view, a virtual machine is described by its interface. The interface of the virtual machine may be formalized as in [3]:

Let S be a finite set of data types.

Let F be a finite set of function types called primitive functions of the virtual machine.

The function

$$fct : F \rightarrow S^*$$

associates its arguments and result type sequence to each primitive function:

$$fct(f) = (s_1, \ldots, s_{n+1})$$

This interface is used by means of a language. This language allows building programs (composed functions) using functions composition operations, each function in such a composition being a primitive function or a composed one.

2.2 Implementation model

Let V be a finite set of carrier sets, instances ale types in S, so that for each $s \in S$ the carrier set $s^V \in V$ is associated.

Let P be a finite set of carrier function types so that for each $f \in F$ the set $f^P \in P^*$ is associated.

The set f^P is the set of functions whose composition, based on an algorithm, represents the implementation of f. This represents the support, in API acception, for execution of the programs written in the virtual machine language.

A typical interpreter implements "fetch-decode-execute" cycle for each operation in F. The " execute" component of this cycle means to apply the composition (algorithm) defined for the operation f over the functions in f^P.

The virtual machine has an internal events triggering mechanism. These events (exceptions) are triggered based on a logical predicates set implemented at the platform level.

Let E be the set of these predicates.

The virtual machine is defined by the following set of sets

$$VM = \{S, F, V, P, E\}.$$

The set E is refined into three categories.

The first category defines logic restrictions on the values from sets in V. These are filters on the possible values sets, s_i^V, for input data types. Let E_v be the set of these predicates.

The second category are the predicates specific to a certain executions model. In collaborative systems the model is that of parallel executions and concurrent accesses to shared resources. In order to implement this model, predicates for mutual exclusion will be specified. Let E_x be the set of these predicates.

The third category imposes restrictions on the results returned by different compositions of the functions in f. Let E_r be the set of these predicates.

A refinement of the virtual machine definition for the collaborative virtual machine is the set

$$VM^{\tilde{}} = \{S, F, V, P, E_v, E_x, E_r\}.$$

2.3 Open collaborative distributed system model

A collaborative distributed system is a collection of distributed processing elements that execute a common activity. Each processing element, k, can execute its own set of functions, denoted by I^k. This is a subset of the virtual machine functions set $I^k \subseteq F$. The intersection of all I^k sets , denoted by $\bigcap_k I^k$, contains a not empty collection of functions. These are the functions specific to the collaborative work.

Open systems connect to the external environment by means of specialized interfaces. The functions accessed by the interfaces are also part of the virtual machine functions set. We denote by I_p^k the internal functions set, by I_{ei}^k the input interface functions set and by I_{eo}^k the output interface functions set. The following is true:

$$I^k = I_p^k \cup I_{ei}^k \cup I_{eo}^k.$$

As regarding the interactive activities in collaborative systems, we can divide the interface functions into two categories.

The first category is represented by display operations for different information types in the system.These represent the output interface functions set, denoted by I_{eo}^k . It has three components that refer to agent choreography in the collaborative system, to coordination and synchronization in the system activity and to the agent view of the collaborative activity made of all information necessary to the agent in order to play its role in the system choreography. The set I_{eo}^k is composed of three subsets corresponding to the three components, denoted by I_{ou}^k , I_{os}^k and I_{ov}^k respectively, so that

$$I_{eo}^k = I_{ou}^k \cup I_{os}^k \cup I_{ov}^k.$$

The second category has two components corresponding to the operations activated by the user. We denote by I_u^k the set of the application operations accessible by means of the interface controls and with I_s^k the set of system operations accessible at the user interface level. The following is true:

$$I_{ei}^k = I_u^k \cup I_s^k.$$

The operation in these sets are specific to each agent type but are different form the operations in the set I_p^k , and have, in general, non in empty intersections with it. The functions in these intersections correspond to interface operations implied in the collaborative activity among the human agents and are a subset of the functions set $\bigcap_k I^k$.

In the collaborative activity each processing element executes an activity described, using the collaborative virtual machine language, in a script.

An agent, in a large acception of semnificative functional element of a system, is processing element executing an activity. Agents communicate by using a shared communication space composed by information units. Each activity operates with its own collection of information units called execution context. The shared communication space is the intersection of the execution contexts of the active agents in the system. The predicates in the set E_x are associated to these intersections and are used to manage locks over the information units.

3. Open distributed collaborative framework based on the formal model

The model is the system abstraction at design time. The system is characterized by a high flexibility if the model is available at runtime because, changing the model, the system can be dynamically redesigned.

Based on the formal model defined previously we propose a framework for open distributed collaborative systems.

The framework specifications contain data specifications, services specifications and inter-components communication model specifications.

3.1 Data specifications. Data and metadata

DataItemType represents an element $s \in S$ and contains the type identifier and the source for building the carrier set, s^V, corresponding to the type.

DataItem is an instance d of the type s ,

$$d \in s^V$$

and represents an interface to an information unit.

Let F be the set of function symbols associated to the operations of the language.

The function

$$Intp : F \to P^*$$

associates to each function in F the set of functions in P implied in its execution.

AgentType is characterized by a set of functionalities realized by means of an interpreter. This set of functionalities is represented by $I^k \subseteq F$ being a subset of virtual machine operations set.

The interpreter of an *AgentType* is an algorithmic composition of the functions from the co-domain of

$$Intp^k : I^k \rightarrow P^*$$

If the agent is connected to the external environment the interpreter has three components that correspond to the sets:

$$Intp^k_p : I^k_p \rightarrow P^*; Intp^k_{ei} : I^k_{ei} \rightarrow P^*;\ \ Intp^k_{eo} : I^k_{eo} \rightarrow P^*\ .$$

If the agent is connected through interactive interface, the interpreter can be refined to six components corresponding to the following sets:

$$Intp^k_p : I^k_p \rightarrow P^*;\ Intp^k_u : I^k_u \rightarrow P^*;\ Intp^k_s : I^k_s \rightarrow P^*;$$

$$Intp^k_{ou} : I^k_{ou} \rightarrow P^*;\ \ Intp^k_{os} : I^k_{os} \rightarrow P^*;\ Intp^k_{ov} : I^k_{ov} \rightarrow P^*.$$

ActivityType defines an activity that must be executed by the agent. The execution needs a certain interpreter, corresponding to the set I^k . *ActivityType* contains an algorithmic composition of some operations from the set I^k , called *script*, the data types associated to these operations, i.e.$DI \subseteq S$ as input data and $DO \subseteq S$ as output data. *ActivityType* also contains the interface type with the external environment and script contains three subcomponents dedicated to output activities, to synchronous executions and to input events handling:*script$_o$*,*script$_p$* and *script$_i$* respectively. In conclusion, *ActivityType* is represented by: $A = \{InterfaceType, DI, DO, script_p\}$.

Agent is an instance of *AgentType* .

InterfaceType specifies an interface type characterized by two sets of operations, operations at the application level and operations at the system level, and by three sets of displayable data types. An interface type contains two algorithmic compositions of functions from I^k_u and from I^k_s respectively, for each category of interface activities responsible with events handling and an algorithmic composition of functions from I^k_{eo} , corresponding to the output activities. *InterfaceType* is represented by the set $B = \{script_o, script_i\}$ which can be refined to $B^{\tilde{}} = \{scripto, \{script_{iu}, script_{is}\}\}$

In the case of interactive interfaces dedicated to the human user, a new refinement of the set B is introduced, according to the structure defined for this interface:

$$C = \{\{script_{ou}, script_{os}, script_{ov}\}, \{script_{iu}, script_{is}\}\},$$

where $script_o$ is refined to $\{script_{ou}, script_{os}, script_{ov}\}$ and $script_i$ is refined to $\{script_{iu}, script_{is}\}$. The element $script_{ou}$ is the script that defines the operations for displaying the choreography of the role to which the user is attached, $script_{os}$ defines the operations for displaying the information about the system state and the way in which the user is reported to them according to the role he plays in the collaboration, and $script_{ov}$ defines the operations for displaying the application level information that must be known by the user in order to realize his task. The input operations connected to the collaborative application logic are defined in $script_{iu}$ and those corresponding to system operations available to the user are defined in $script_{ov}$.

An *InterfaceType* is an instance of B type or of its refinements $B^{\tilde{}}$ or C.

3.2 Services specifications

A flexible collaborative platform has two distinct functions: the function of modelling and dynamic building of application and virtual machine components and the function of executing the collaborative application.

According to this, the framework model has two components. The component responsible to data creation, which hides the data creation details, is realized using dynamic factories. Dynamic factories are components of the *modelling and dynamic building service* and are responsible with data instantiation in a flexible manner.

The component dedicated to collaborative executions represents the underlying support for collaborative application execution. It has four services specific to a typical runtime support.

These services are *executions negotiation service, execution contexts management service, event service* and *predicate service.*

The *modelling and dynamic building service* manages the set of types S, F, Ev, Er, A and their corresponding refinements and offers operations for creation and for elimination types from the system. The service is also responsible to entities dynamic building based on these type definitions. The building operation uses carrier sets V and P. A subcomponent of this service is that which models and builds different types of the interfaces to the external environment. It is based on the model of the interfaces with the external environment and uses the set B and its refinements.

Execution contexts management service dynamically manages the sets of instances s_k^V of types $s_k \in S$. Execution contexts contain sets of information units accessed and operated in common by more agents. The service keeps the consistence of these information units.

The instances $\{s_k^V \ , \ s_k \in S \mid \forall i \neq k \ , \ s_k^V \cap s_i^V = \theta\}$ can be transfered to the agent and locally modified without consistency implications. This will raise the performance of the executions.

The instances $\{s_k^V \ , \ s_k \in S \mid \exists i \neq k \ , \ s_k^V \cap s_i^V \neq \theta\}$ need solutions for consistency because are shared by more agents in the system. These instances belong to the shared space and are represented by the predicate set E_x whose content is dynamically modified during the collaborative activity evolution.

Executions negotiation service manages requests and offers for executions and is responsible with distribution and coordination of the activities defined in the collaborative application. It also contains the mechanism for adaptive integration and re-adaptation of agents based on a negotiation process of the functional offers and needs[10].

Executions negotiation service initiates agents creation and activation, agent local machine services integration and dynamic change of the agent according to dynamic change of the agent type. Agents are created by downloading the corresponding algorithmic compositions of the functions from the sets defined by

$$Intp^k : I^k \to P^*,$$

composition that represents a component of the interpreter. Another component of the interpreter is added in order to support local services access, services offered by the agent and adaptive integrated in the system when the agent is attached.

Event service has the classic functions of a service of this type. It intermediates the asynchronous inter-agent communication.

Predicate service implements the exception mechanism and has three predicate types formally defined by E_v , E_x and E_r. The component corresponding to the predicates in the set E_x interfaces with *execution contexts management service* in order to manage the consistency for instances in the set

$$\{s_k^V \ , \ s_k \in S \mid \exists i \neq k \ , \ s_k^V \cap s_i^V \neq \theta\}$$

The component E_v acts as an input filter for operations execution in the system and the component E_r acts post-operation for results validation.

3.3 Inter-components communication model

The dynamic view of the system model contains the mechanisms by which dynamic modelling, collaborative executions and adaptive integration are realized.

The modelling and meta-modelling component contains the mechanisms for *types dynamic integration*, for *agents adaptive integration* and for *dynamic* *predicates*, defining internal system restrictions, *integration.* All the mechanisms for dynamic and adaptive integration use the facilities offered by the *dynamic modelling and building service* and the interface specifications that are designed for integration and are present in the interface of the service that manages the integrated entity.

The component for collaborative application execution contains two mechanisms necessary for a collaborative application execution over a distributed system with an architecture in which communication are based on shared information space. These are the *executions request and negotiation mechanism* and the *mechanism for shared information consistency.* Each mechanism is based on a collaboration between two or more services. The *mechanism for executions request and negotiation* is realized by the *executions negotiation service* in collaboration with the *agents* in the system. The mechanism for shared information consistency is realized by the *execution contexts management service* in collaboration with *predicate service.*

4. Dynamic Development and Adaptive Integration

Dynamic development of collaborative applications using the proposed framework has five phases. The first two phases are dedicated to initial definition of the collaboration.

In the first phase, the initial functionality of the virtual machine is established by defining the sets of types S, F, V, P, E and the correspondences fct.

In the second phase, the initial functionality of the application is established by defining the sets of types A, B , the correspondences $Intp$ and their refinements.

The next three phases represent the dynamic component of the collaborative application development because are realized while the application is executed.

The third phase modifies the virtual machine functionality. If the modification is realized at the information level, a new element in the carrier set V is created. If the modification is realized at the operations level, a new element in the carrier set P is created. The new element is registered at the appropriate dynamic factory and the executions negotiation service is notified.

In the fourth phase, application functionalities are changed. This is realized by creating an element in the set A, an element in the set B and, if necessary, an element in the carrier set $Intp$. These elements are registered to the appropriate dynamic factory and the executions negotiation service is notified.

The fifth phase is represented by the dynamic integration of changes and is realized after each phase that implies a change. The dynamic integration operations are realized automatically based on the mechanisms implemented in the collection of the framework services. Following the change notifications, the executions negotiation service starts the integration mechanism that collaborates with the agents in order to identify the integration moment. Integration of an activity type can be executed only if the activity is not currently executed at any agent. Similarly, the integration of a data type modification can be executed only if that type is not currently used in the system.

The adaptive integration of the agents in the system has three phases.

In the first phase a new agent is created by the executions negotiation service. The functionality of this new agent is initialized according to the *AgentType*.

This new agent is created in a context that can have a specific available functionality. In the second phase this functionality can be partially integrated in the system using the adaptive integration mechanism based on negotiation [10]. The results of the negotiation process will be new functionalities at the virtual machine level.

The third phase modify the initial functionality of the virtual machine according to the negotiation results. These modifications are applied to the sets of types S, F, V, P, E and to the sets of correspondences fct and $Intp$.

5. Conclusions

This paper referred to the domain of adaptive collaborative environments and introduced a simple mathematical model, based on the theory of sets[2] and a framework that uses this model as support for flexibility.

Modelling abilities offered by the framework are the basis for collaborative applications dynamic development and for adaptive integration of agents in the collaborative application.

This framework allows both dynamic modelling of collaborative applications and dynamic modelling of support API in order to allow any application definitions. The framework contains four typical platform services as support for collaborative executions and a service based on metamodelling that offers possibilities for modelling and dynamic building of system entities.

When integrating a new agent, the system functionality evolves by extending the system interface semantics. As the application develops, the agent may “ learn” new functions, according to system needs. It also may “ remember” what it already knew when entering the system. The framework has the possibility to develop a “ socializing” mechanism. The agents must adapt their functionality to the agents society needs. This mechanism offers to the agents a set of facilities to negotiate the accepted functions and the collaboration rules so that we can tell that the agents integration process take into account their “ personality” and they can also readapt in the future.

This framework represents a resultant of different existing specifications for collaborative systems from the flexibility point of view because it allows dynamic changes on all dimensions of the application – virtual machine assembly. Its flexibility refers both to the dynamic change of the virtual machine API and of the collaborative application definitions and to the possibility of new agents adaptive integration at runtime.

References

[1] Aissi Selim, Malu Pallavi, Srinivasan Krishnamurthy, *E-Business Process Modelling: The Next Big Step*, Intel Labs, may 2002 IEEE.

[2] Bourbaki N., *Elements of Mathematics Theory of Sets*, Addison_Wesley, 1968 (translated form French).

[3] Broy Manfred, *Mathematical System Models as a Basis of Software Engineering*, Computer Science Today 1995.

[4] David L. Martin, Cheyer Adam J., Moran Douglas B., *The Open Agent Architecture: A Framework for Building Distributed Software Systems*, 1998.

[5] Grundy J.C., Apperley M.D., Hosking J.G., Mugridge W.B., *A Descentralized Architecture for Software Process Modelling and Enactement*, IEEE INTERNET COMPUTING, sept.-oct. 1998.

[6] Huhns Michael N., Singh Munindar P., *Cognitive Agents, IEEE INTERNET COMPUTING*, november-december 1998.

[7] Huhns Michael N., Singh Munindar P., *Workflow Agents, IEEE INTERNET COMPUTING*, july-august 1998.

[8] *Living systems R transforming markets – living markets* - Technical White Paper - 2001

[9] *OAA specifications, v2.2* http://www.ai.sri.com/oaa2.

[10] C Mîndruta, *A meta-modelling based support for adaptive integration in agent-based systems.* The 10th Annual Concurrent Engeneering Conference'2003, apr. 14-16 2003, Plymouth.

[11] C. Mîndruta (2003) *Contributions to the collaborative applications development using distributed systems*, PhD. Thesis, to be published.

"Ovidius" University of Constantza,
Department of Mathematics & Informatics ,
Bld. Mamaia 124,
8700 Constantza,
Romania
e-mail: cmandruta@univ-ovidius.ro