On a Framework for Modeling Collaborative Distributed Systems

Cristina Mindruţă

Department of Informatics and Numeric Methods, OVIDIUS University of Constanţa, Faculty of Mathematics and Informatics, B.dl Mamaia no.124, 900257 Constanţa, Romania, Phone: 0040241618070, E-Mail: cmandruta@univ-ovidius.ro

Abstract Metamodeling is a powerful tool to manage the rapid increasing complexity of distributed computing systems. We present a formal framework, based on a glass-box view of what, at a certain abstraction level can be considered to be the “atomic” component of a complex computing system, for modeling collaboration in distributed systems. We also discuss a possible application in designing specifications for a container-based platform for developing collaborative applications. Based on the flexibility and on adaptive and reflective features of the used metamodel, the formal definitions introduced can be used in the implementation of a framework for adaptive and reflective collaborative distributed applications. This work will be continued with an implementation using the new facilities introduced by EJB 3.0.

Keywords: metamodeling, reflection, adaptive system, collaborative distributed application.

1. INTRODUCTION

Main research efforts in the field of collaborative systems follow the current trends in distributed systems and mobile computing for defining flexible, adaptive and reflective systems.

Different solutions have been proposed as responses to such requests. The solution may refer to the partitioning of functions, components having associated functions of a typical distributed synchronous collaborative system. The context of use, actors, task tree, workflow and artifacts of collaboration activity, are considered main components of a collaborative system. What is essential here is the idea of the separation of concerns, this being the first step in designing flexible systems, flexibility being also the basis for adaptive behavior. In the same direction, in [1] the idea of separating data and control is proved as the support for adaptable consistency protocols in collaborative systems.

As regards flexibility, the authors in [2] focus on collaborative systems architecture and propose a model for dynamic collaboration architectures. They define a user level function that dynamically changes the mapping between user interface and program components so that dynamically changing between centralized, replicated and hybrid collaboration architectures can be made. COCA (Collaborative Objects Coordination Architecture) [3] is another example of a flexible architecture, implemented in a virtual machine, cocavm, which supports roles and dynamic policies for collaboration.

The complexity of distributed applications has a high growing rate due to new technological developments, interfacing devices diversity and user requirements. Dynamic changes support is necessary in building adaptive systems and this is what is needed in distributed applications that must support mobile computing. In our paper we will focus on collaborative distributed applications.

General ways to manage systems complexity are based on introducing new abstraction levels. We present here a conceptual modeling framework, a metamodel, for distributed systems and apply it to collaborative applications needs, in order to build fundamental specifications for a container-based execution platform for such applications.

The meta-model gives set representations for the type system of software platforms. Our notion of type system of software platform inherits from the concept of type system used in [4] and extends it to implementation types. The representation of the systems in terms of types offers a basis for a reflective view over the modeled system. Reflection has the benefit of dynamic changes control and has been exploited on different components of the systems. The component we refer to is the type system, reflection allowing types changes control in collaborative distributed systems.

We use a previously proposed [5] core mathematical metamodel, based on the naive theory of sets. This metamodel is the model of a generic computing system, the virtual machine, considered the “atomic” component for building complex computing systems. At a high level of abstraction, a virtual machine is an assembly of two components: API and runtime environment. This definition was refined and developed, in terms of types, giving us a glass-box view of the virtual machine. API is the set of function types and data types used in the applications developed for the virtual machine. The runtime environment is the set of services that support the applications execution.
The next two sections of the paper contain a review of the metamodel and its application in modeling collaborative distributed systems in terms of types. Section IV contains basic concepts for collaborative systems and the way they are modeled in this formal framework in order to extract specifications for a collaborative container. Section V shows the starting idea for implementing these specifications using new facilities introduced by EJB 3.0. The last section presents the conclusions and the intentions for future work on this respect.

II. VIRTUAL MACHINE META-MODEL

In this section we review the formal model introduced in [5]. It contains the interface, the implementation and the services models.

The interface of the virtual machine is formalized by the set of sets \( I_e = \{ S, F, TE, AE \} \) and by the set of correspondences \( I_e = \{ fct, evt \} \), where \( S \) is a finite set of types, representing the interface data types, \( F \) is a finite set of types, representing the interface functions, \( TE \) is a finite set of types, representing the interface event types that can be generated by the virtual machine and \( AE \) is a finite set of types, representing the interface external events that can be received by the virtual machine.

The function \( fct:F \rightarrow P(S) \) associates to each primitive function, the set of its arguments types and result type.
The function \( evt:F \rightarrow P(TE) \) associates its event types set to each primitive function.

The implementation model is a key element because it connects interface to underlying level, thus giving the “glass box view” characteristic to our model.
The implementation model identifies underlying level types as implementation types and maps interfaces types into them. It also captures type relations for events handling and relations between event types.

Let \( V \) be a finite set of carrier types, sources of types in \( S \).
The function \( Srs:S \rightarrow P(V) \) associates the carrier set \( s' \in P(V) \) to each \( s \in S \), where \( s' = \{ \delta_1, \delta_2, \ldots, \delta_t \} \) is the set of data types, \( \delta \epsilon V \), used to create the definition of the type \( s \).

Let \( P \) be a finite set of carrier function types. The function \( Intp:F \rightarrow P(P) \) associates the set \( f' \in P(P) \) to each \( f \in F \). The set \( f' \) is the set of functions in the implementation of \( f \) in a language defined over the alphabet \( P \).

The sets \( P \) and \( V \) are related by a function similar to \( fct \)
\[ fct*:P \rightarrow P(V). \]

In our approach, the internal events of a virtual machine, also called exceptions, are generated based on a logical predicate set implemented at the platform level and they result from the current activity on the virtual machine.

Let \( SE \) be the set of internal event types.

The function \( Trigg:SE \rightarrow P(S) \times P(P) \) associates the sets \( e_1 \in P(S) \) and \( e_2 \in P(P) \) to each event type \( e \in SE \).
The set \( e_1 \) is the set of data types implied in the expression representing the event \( e \) triggering condition (predicate).
The set \( e_2 \) is the set of function in the implementation of the trigger actions, for the event of type \( e \).

The sets \( P \) and \( SE \) are related by a function similar to \( evt \)
\[ evt*:P \rightarrow P(SE) \]. This function associates to each carrier function in \( P \) the set of internal events that its execution can generate. The types of these events are a subset of \( SE \) and their corresponding values for \( Trigg \) are in the range \( \emptyset \times P \).

Asynchronous events (interrupts) are generated by entities in the virtual machine’s environment and are not synchronous with the current activity on the virtual machine. As specified in the interface model, \( AE \) is the set of external event types and is a component part of the virtual machine interface.

The function \( Rut:AE \rightarrow P(P) \) associates the set of the functions in the implementation of the internal event handler for the event of type \( i, i' \epsilon P(P), \) to each event type \( i \epsilon AE \).

The function \( Th:SE \cup AE \rightarrow F(TE) \cup \{ \emptyset \} \) associates the set \( t \in F(TE) \cup \{ \emptyset \} \) to each \( e \in SE \), or associates the set \( t \in F(TE) \cup \{ \emptyset \} \) to each \( e \in AE \).

The sets \( f' \) and \( f \) are the sets of events generated in the interface of the virtual machine as consequences of the event \( e \) and \( i \), respectively.

Consequently, the static metamodel of the virtual machine is defined as the 7-tuple of finite sets:

\[ VM' = \{ S, F, TE, V, P, SE, AE \} \]

and the set of correspondences:

\[ VM' = \{ fct, evt, Srs, Intp, Trigg, Rut, Th \} \]

From the type system point of view, the dynamic aspect of the virtual machine refers to the collection of specific services and mechanisms, which realize the abstractions of virtual resources supporting applications execution.

According to the fundamental models of computing systems, we consider that the core services are: processing service(PS), memory service(MS), event service(ES) and predicate service(TS). The reflective feature is obtained by extending the system with a specific service, modeling and

1 \( F(TE) \) is a division of \( TE \).
**dynamic building service**, denoted by $DSv$ and responsible to type creation and instantiation at runtime.

We denote by $Sv$ defined as

$$Sv = \{ PSv, MSv, ESv, TSv, DSv \}$$

the set of virtual machine core services types.

The functions of each service relate to the previously defined model as follows.

Let $Sv$ be the set of services. The correspondence:

$$dy: Sv \rightarrow P(VM^2) \times P(VM^3)$$

associates to each service $s_i \in Sv$ the corresponding sets of entities in the static metamodel.

The values of the function $dy$ applied to the core services are:

$$\begin{align*}
   dy(PSv) = \{ \{ S, F, P \}, \{ \text{fct, Intp} \} \} \\
   dy(MSv) = \{ \{ S, V \}, \{ \text{Srs} \} \} \\
   dy(ESv) = \{ \{ AE, TE, P \}, \{ \text{Rut, Th} \} \} \\
   dy(TSv) = \{ \{ S, F, SE, TE, P \}, \{ \text{evt, Trigg, Th} \} \} \\
   dy(DSv) = \{ \{ S, F, TE, V, P, SE, AE \}, \{ \text{fct, evt, Srs, Intp, Trigg, Rut, Th} \} \}
\end{align*}$$

The complete core formal metamodel of the virtual machine in terms of types is:

$$VM = \{ VM^2, VM^3, Sv, dy \}$$

### III. COLLABORATIVE DISTRIBUTED SYSTEMS

Based on the recursive definition of multiple virtual machines systems in multi-layered architectures defined in [5], the core formal model can be used to define, in terms of types, complex computing systems. As a virtual machine on the level $k+1$, a distributed system is realized with a collection of underlying, on the level $k$, virtual machines managed by specific services that may be implemented with different distribution degrees, from centralized to fully distributed.

The distributed virtual machine adds its specific types to the union of the type systems of the virtual machines in the net. These types are added in the interface of the level $k+1$. We denote them by $dSv_{k+1}$ for data types and by $dFv_{k+1}$ for function types. Consequently, data types $S_{k+1}$ and function types $F_{k+1}$, in the type system of the distributed virtual machine, are represented by:

$$\begin{align*}
   &s_{k+1} \leftarrow \bigcup_{i=1}^{n} s_{k}[i] \cup dSv_{k+1} \\
   &f_{k+1} \leftarrow \bigcup_{i=1}^{n} f_{k}[i] \cup dFv_{k+1}
\end{align*}$$

The distributed virtual machine hides data and work distribution and also communication and synchronization operations. In order to transparently manage these operations, the distributed virtual machine needs specific mechanisms, implemented by its services using underlying types. The set $Sv$ is extended with new service types responsible to communication and synchronization operations, to work and data distribution, to lifecycle. The model we use allows us to represent an extended services collection. We generally denote these distributed services by $DsSv$ and represent the service types set of the distributed virtual machine by:

$$Sv = \{ PSv, MSv, ESv, TSv, DsSv, \ldots, DsSvn \}.$$ 

A significant distributed system is the collaborative system. According to our formalism, it is a special type of distributed virtual machine. Collaborative systems have specific operations and services in order to manage collaboration. The model of the distributed virtual machine extended with these specific data, functions and services types will represent the model of the collaborative virtual machine.

The key service is **predicate service**, which implements collaboration rules. As regards the formal definitions, collaboration rules are part of the $SE_{k+1}/TE_{k}$ set. In order to model a collaborative virtual machine, the set $SE_{k+1}$ is refined into three subsets.

The first subset defines logic restrictions on the values of data with types in sets in $V_{k+1}$, where $\nu_{k+1} - \bigcup_{i=1}^{n} s_{k}[i]$.

These are filters on the value set for some data types. Let $SE_{k+1}$ be the set of these internal event types and let $Trigg^k: SE_{k+1} \rightarrow P(V_{k+1}) \times P(P_{k+1})$ be the restriction of the function $Trigg$ to the domain $SE_{k+1}$; $\nu_{k+1} - \bigcup_{i=1}^{n} f_{k}[i]$.

The second subset defines logic restrictions specific to a certain model for collaborative executions. Particular collaboration models will specify the global common data set $S_{k}[c]$, operation set $F_{k}[c]$ and event sets $TE_{k}[c]$ and $AE_{k}[c]$ as subsets of the corresponding sets in $VM^2[i]$, where $i$ identifies a virtual machine implied in the collaboration.

Fig. 1. represents two virtual machines with their interface types and an instance of event-based collaboration specific relations, with a collaboration event generated by $VM_{k}[i]$ and received and processed by $VM_{j}[j]$.
The separation of collaboration types and relations represented in this figure suggests separation between container definitions and business definitions in terms of types. The communication between virtual machines in collaborative systems is generally realized using a shared memory model, according to the blackboard architecture. It is assisted by notifications about changes in blackboard. A call of a function of a type in \( F[i] \) of \( VM_i \) may generate a change in the blackboard content, change that will be notified by an event, with the type \( t e T E[c] \), generated in the interface of \( VM_i \). This event is received by \( VM_j \) as an interface event of a type in \( AE[c] \). It is processed by \( VM_j \) generating an event of a type in its \( TE[c] \) set, which is asynchronously communicated to the business interface of \( VM_j \) as an event of a type in \( AE[j] \).

\[ S[c], F[c], TE[c] \] and \( AE[c] \) are the subsets of the virtual machine definition on which the general model, of parallel executions and concurrent accesses to shared resources, is applied, so they must be kept synchronized. The coupling degree, from tight to loose coupling, depends on the configuration of these sets. Different access rights of the roles also can be established, for example to implement a master/slave coupling model, when a single role is allowed to change the common data set.

For implementing the model of parallel executions and concurrent accesses to shared resources, predicates for mutual exclusion will be specified. Supplementary, predicates for collaboration rules must be specified too. Let \( SE_{k+1} \) be the set of the internal events types generated by these predicates and let \( Trigg: SE_{k+1} \rightarrow P(S[c]) \) be the restriction of the function \( Trigg \) to \( SE_{k+1} \).

The third subset represents restrictions on the results returned by possible compositions of the functions in \( P \). Let \( SE_{k+1} \) be the set of these internal event types and let \( Trigg: SE_{k+1} \rightarrow P(V[c]) \) be the correspondent restriction of the function \( Trigg \).

Two supplementary conditions also exist:

\[ SE_{k+1} = SE_{k+1} \cup SE_{k+1} \cap SE_{k+1} \cap SE_{k+1} = \emptyset \]

If \( SE_{k+1} \subseteq SE_{k+1} \), the model is that of atomic operations on input data, which include both data validation from the content type point of view and from the consistency point of view, possibly extended with data transfer according to consistency requests. If \( SE_{k+1} \cap SE_{k+1} = \emptyset \) the model is that of different atomic operations, one for input data validation from the content type point of view and the second for validation from the consistency point of view, possibly extended with data transfer according to consistency requests. In the second case we have a more fine level of atomicity that allows a better performance.

The same discussion, about the relation between \( SE_{k+1} \) and \( SE_{k+1} \), is true for output data too.

As our platform is reflexive and consequently adaptive, rules can also be changed\(^3\). Changing rules while they are applied is a dangerous operation. That is why, as we have discussed in [6], a negotiation mechanism is used when changing a type. Negotiation with a service is a relation between the modeling and dynamic building service and the other service of the execution environment. This operation is executed at runtime in order to allow dynamic integration of types as support for adaptive behavior.

### IV. COLLABORATIVE SPECIFICATIONS

A collaborative application using a distributed system uses the collaboration specific types in the sets \( S[c], F[c], TE[c], AE[c] \) and \( SE_{k+1} \). These types may be implemented either by the application programmer or declared by the programmer and implemented by one or more services of the distributed system. In the second case, a container is built at the level of the distributed system. This container externalizes collaborative specific operations and hides their implementation. Consequently, we get a collaborative distributed virtual machine over which we can run collaborative applications.

Collaborative systems are systems that support goal-oriented working in collaboration to carry out joint projects that are characterized by the need of communication, planning and coordinating tasks, of monitoring project progress and of cooperation. A collaborative activity is realized by a team that can be composed of more groups. Teams are task-oriented and operate in distributed organizational settings.

The concept of group awareness refers to the fact that each member of a group is aware of actions, plans, goals and activities of the other members in the same group. Each collaborative entity is attached to a role. All entities express need for information sharing, task scheduling, role taking, synchronization and resource allocation. In [7] a more specific concept, that of activity awareness, was defined. It integrates not only activity definitions, data and events, but also tools, knowledge and expectations, attitudes and evaluation criteria. This extension of the concept applies to collaborative application design and is not our concern, as we are interested especially in the relation between application and its execution platform. As regards the concepts defined in the same paper, we are still interested in the concepts of group cohesion based on the dimension of the default global context. The default global context is the set of data, operations and rules implemented by the underlying system and which can be modeled by our formalism in a simple and straightforward manner. Our formalism can also capture the process of grounding.

\[^3\] In order to eliminate chaos, at least one rule must exist and be immutable.
defined there as the continuous adaptation to the common ground, where common ground represents the joint awareness two communicating entities share. In other words, common ground is the set of elements whose complete semantic is shared by the two entities and the grounding behavior is the process of maintaining common ground. The organization of common grounds mimics the organizational model of each particular system.

An example of a collaborative container component can be considered the VRUI [8] (Virtual Reality User Interface), a library that implements collaborative features. It has a base class that implements support for collaborative work by automatically distributing events from local user interactions to all remote partners. This operation can be enabled or disabled according to the local needs and to the collaboration model.

In our work we tried to connect some research ideas to an abstract model.

A collaborative virtual machine allows collaborative application definitions and offers a specific support for collaborative application execution based on cooperative process interaction.

Support for collaborative application definition means the possibility to define the collaboration: the common goal, the roles, the activities and the correspondences role-activity, i.e. role implementation. From the proposed framework point of view, the definition of a role means specifying subsets of types in the virtual machine interface, i.e. subsets of $S_{k}$, $F_{k}$, $TE_{k}$, $AE_{k}$, and also their type relations as the restrictions of the correspondences $fct$, $evt$ and $Th$ to these subsets. The implementation of a role extends the type definitions to the subsets of the implementation metamodel and their correspondences. The previous definitions represent the type system of the role and can be viewed as a set-based image of the role.

A support for collaborative application execution contains operations specific to cooperative processes interaction and means to solve concurrency problems for shared resources. In this respect, the set-based image of a role can be made available to all the members of a group as part of group awareness information. The set-based image of default global context can be also defined in the same manner. As part of the support for grounding behavior, set operations can be applied at runtime to process the set-based images of communicating entities in order to dynamically build type definitions for their common ground.

We also can use the core metamodel and its refinement in order to define the interface of a collaborative container. Basically, it contains the methods to define the subsets of sets in $VM^{k}$ and $VM^{k+1}$, containing the collaboration specific elements ($S[c]$, $F[c]$, $TE[c]$, $AE[c]$ on the level $k$ and $SE[c]$ on the level $k+1$) and the correspondent elements in the mappings $VM^{k}$ and $VM^{k+1}$. These methods are integrated in an organizational model of the collaborative team made of groups of collaborators attached to different roles. In this respect, the interface of the collaborative container also contains methods for specifying the role assumed and the group to which the connected component is attached.

As the system is intended to be adaptive, these operations are available at runtime too, but must be backedgrounded with an implementation that offers all consistency requirements, from classic information consistency to operating consistency. The later refer to the aid in implementation that allows dynamic type changes management. More precisely, reflective collaborative systems can be built by adding the modeling and dynamic building service, $DSv$ to a collaborative virtual machine. Having the model, in terms of types, of collaborative computing systems, the implementation of a transfer function for a dynamic changing can be supported by adding to it a part which implements the change methodology defined in [6].

V. EXPERIMENTAL DIRECTIONS

Based on the possible analogy that, in a restricted sense, a collaborative container is to a distributed application container what a monitor (as mechanism for mutual exclusion) is to a classical object, we intend to define and implement collaborative features in extension to the EJB 3.0 container.

The specifications for EJB 3.0., in [9], introduce some important facilities and more flexibility in working with EJBs. These facilities are programmer-defined interceptors and the dependencies injection mechanism.

We also can use the core metamodel and its refinement in order to define the interface of a collaborative container. Basically, it contains the methods to define the subsets of sets in $VM^{k}$ and $VM^{k+1}$, containing the collaboration specific elements ($S[c]$, $F[c]$, $TE[c]$, $AE[c]$ on the level $k$ and $SE[c]$ on the level $k+1$) and the correspondent elements in the mappings $VM^{k}$ and $VM^{k+1}$. These methods are integrated in an organizational model of the collaborative team made of groups of collaborators attached to different roles. In this respect, the interface of the collaborative container also contains methods for specifying the role assumed and the group to which the connected component is attached.

As the system is intended to be adaptive, these operations are available at runtime too, but must be backedgrounded with an implementation that offers all consistency requirements, from classic information consistency to operating consistency. The later refer to the aid in implementation that allows dynamic type changes management. More precisely, reflective collaborative systems can be built by adding the modeling and dynamic building service, $DSv$ to a collaborative virtual machine. Having the model, in terms of types, of collaborative computing systems, the implementation of a transfer function for a dynamic changing can be supported by adding to it a part which implements the change methodology defined in [6].

V. EXPERIMENTAL DIRECTIONS

Based on the possible analogy that, in a restricted sense, a collaborative container is to a distributed application container what a monitor (as mechanism for mutual exclusion) is to a classical object, we intend to define and implement collaborative features in extension to the EJB 3.0 container.

The specifications for EJB 3.0., in [9], introduce some important facilities and more flexibility in working with EJBs. These facilities are programmer-defined interceptors and the dependencies injection mechanism.

We also can use the core metamodel and its refinement in order to define the interface of a collaborative container. Basically, it contains the methods to define the subsets of sets in $VM^{k}$ and $VM^{k+1}$, containing the collaboration specific elements ($S[c]$, $F[c]$, $TE[c]$, $AE[c]$ on the level $k$ and $SE[c]$ on the level $k+1$) and the correspondent elements in the mappings $VM^{k}$ and $VM^{k+1}$. These methods are integrated in an organizational model of the collaborative team made of groups of collaborators attached to different roles. In this respect, the interface of the collaborative container also contains methods for specifying the role assumed and the group to which the connected component is attached.

As the system is intended to be adaptive, these operations are available at runtime too, but must be backedgrounded with an implementation that offers all consistency requirements, from classic information consistency to operating consistency. The later refer to the aid in implementation that allows dynamic type changes management. More precisely, reflective collaborative systems can be built by adding the modeling and dynamic building service, $DSv$ to a collaborative virtual machine. Having the model, in terms of types, of collaborative computing systems, the implementation of a transfer function for a dynamic changing can be supported by adding to it a part which implements the change methodology defined in [6].

V. EXPERIMENTAL DIRECTIONS

Based on the possible analogy that, in a restricted sense, a collaborative container is to a distributed application container what a monitor (as mechanism for mutual exclusion) is to a classical object, we intend to define and implement collaborative features in extension to the EJB 3.0 container.

The specifications for EJB 3.0., in [9], introduce some important facilities and more flexibility in working with EJBs. These facilities are programmer-defined interceptors and the dependencies injection mechanism.

We also can use the core metamodel and its refinement in order to define the interface of a collaborative container. Basically, it contains the methods to define the subsets of sets in $VM^{k}$ and $VM^{k+1}$, containing the collaboration specific elements ($S[c]$, $F[c]$, $TE[c]$, $AE[c]$ on the level $k$ and $SE[c]$ on the level $k+1$) and the correspondent elements in the mappings $VM^{k}$ and $VM^{k+1}$. These methods are integrated in an organizational model of the collaborative team made of groups of collaborators attached to different roles. In this respect, the interface of the collaborative container also contains methods for specifying the role assumed and the group to which the connected component is attached.

As the system is intended to be adaptive, these operations are available at runtime too, but must be backedgrounded with an implementation that offers all consistency requirements, from classic information consistency to operating consistency. The later refer to the aid in implementation that allows dynamic type changes management. More precisely, reflective collaborative systems can be built by adding the modeling and dynamic building service, $DSv$ to a collaborative virtual machine. Having the model, in terms of types, of collaborative computing systems, the implementation of a transfer function for a dynamic changing can be supported by adding to it a part which implements the change methodology defined in [6].

V. EXPERIMENTAL DIRECTIONS

Based on the possible analogy that, in a restricted sense, a collaborative container is to a distributed application container what a monitor (as mechanism for mutual exclusion) is to a classical object, we intend to define and implement collaborative features in extension to the EJB 3.0 container.

The specifications for EJB 3.0., in [9], introduce some important facilities and more flexibility in working with EJBs. These facilities are programmer-defined interceptors and the dependencies injection mechanism.

We also can use the core metamodel and its refinement in order to define the interface of a collaborative container. Basically, it contains the methods to define the subsets of sets in $VM^{k}$ and $VM^{k+1}$, containing the collaboration specific elements ($S[c]$, $F[c]$, $TE[c]$, $AE[c]$ on the level $k$ and $SE[c]$ on the level $k+1$) and the correspondent elements in the mappings $VM^{k}$ and $VM^{k+1}$. These methods are integrated in an organizational model of the collaborative team made of groups of collaborators attached to different roles. In this respect, the interface of the collaborative container also contains methods for specifying the role assumed and the group to which the connected component is attached.
events in $AE[c]$. These interceptors must implement collaboration rules, whose types are elements in the set $SE_{k+1}$. It is obvious that each implementation uses subsets of types defined in $S[c]$ and $TE[c]$.

VI. CONCLUSIONS AND FUTURE WORK

A model can have different functions. It can be used to simulate an existing system in order to improve knowledge about it and to lead to the improving of the real system, but models also can be used in order to guide analyzing and design of new systems. The second utility is exploited in software engineering, at the software architecture design level, and allows the understanding of the problem to be solved and of the system main features and also allows decision making before building the system.

In this paper we have referred to this second utility of the model and have introduced a set-based metamodel of a generic atomic computer system as the building block of a framework for modeling collaborative distributed systems. This framework captures the main concepts and characteristics of collaborative distributed systems and reveals basic specifications for a collaborative container.

Such a framework allows the software architect to model different collaborative distributed systems by defining the data types, function types, event types sets and their correspondences.

Based on the possible analogy that, in a restricted sense, a collaborative container is to a distributed application container what a monitor (as mechanism for mutual exclusion) is to a classical object, we intend to define and implement collaborative features to the EJB container.

REFERENCES