Multiagent and Grid Systems – An International Journal
IOS Press

ACVisualizer: A Visualization Tool for Api-Calculus

Shahram Rahimi* and Raheel Ahmad
Department of Computer Science, Southern Illinois University, Carbondale, Illinois 62901
rahimi@cs.siu.edu and rahmad@cs.siu.edu*

Abstract. Process calculi are mathematical tools used for modeling and analyzing the structure and behavior of reactive systems. One such calculus, called APi-calculus (an extension to Pi-calculus), provides support for modeling migration, intelligence, natural grouping and security in agent-based systems [18]. In this article, a visualization tool for the APi-calculus is proposed. Although an invaluable formal method for mobile agent systems, APi-calculus (as other calculi) is quite complex and not well suited to practical applications by itself. Due to the dynamic nature of mobile agents, a graphical program based on the mathematics of the calculus would enhance the use of the calculus and its attraction to the mobile agents industry. The ability to provide instant demonstration of a designed system to the user and the support for verification and validation of such systems specified by the calculus would prove to be a great asset for APi-calculus as well as the mobile agents-based computing in general. This paper presents ACVisualizer, a visualization software for APi-calculus, which provides such high level support for modeling mobile agent systems. Due to the backward compatibility of APi-calculus with Pi-calculus, ACVisualizer can easily be adapted to work with Pi-calculus itself.

Keywords: Process Calculi, Software Agents, System Modeling, Agent-Oriented Software Engineering

Dr. Shahram Rahimi is an assistant professor of computer science at the Southern Illinois University (SIU)-Carbondale. Prior to jointing SIU, he was a visiting assistant professor at the University of Southern Mississippi, where he was a co-director of a NIMA-NURI (DoD) sponsored project involving intelligent database agents for geospatial knowledge integration and management for over three years. Moreover, he has over four years experience as vice president in research with TEMA Engineering Company. Dr. Rahimi has been recently awarded two grants in the area of geospatial data integration and Intelligent Agents. He currently is the director of SIU Software Agents Group (SAG) where faculty and students are involved in several Agent-Based research projects. His main research interests are multi-agent systems, distributed and high performance computing, and Soft Computing. Dr. Rahimi is the editor-in-chief of the International Journal of Computational Intelligence Theory and Practice. He is the associate editor of Informatica Journal for North and South America, editor of International Journal of Computer Science & Applications (IJCSA), a member of the editorial board of Scalable Computing Journal and an editor for Engineering Letters Journal. He has been selected as a chairman and member of the organizing committee for several international conferences.

Mr. Raheel Ahmad is a PhD student in Computer Science at Southern Illinois University. He is currently performing active research in the field of formal methods and software engineering. Mr. Ahmad was awarded a graduate fellowship as an outstanding PhD student on merit basis for the year 2004 - 2005. His research interests have led to several publications in international proceedings and conferences, such as IEEE affiliated conferences, as well as journal articles. He was responsible for the installation and deployment of a high-performance computing cluster for the Computer Science department at SIU. This is intended as a research tool for all the departments in the school that may benefit from the available computing power.

1. Introduction

In order to optimally utilize the available resources for any computing solution, it is prudent, when possible, to employ multiple processes running concurrently, thus dividing the work load. Furthermore, it is necessary that these processes communicate with each other in order to share information and resources. A common and traditional example is a process with multithreading where individual threads are responsible for different tasks, and communicate with each other according to their needs. A more current although very different example is the mobile agent technology.

Mobile agent technology is being heralded as one of the most exciting and enabling technologies in the field of distributed computing in the past few years. Although it may not in general solve any problem that was unsolvable by traditional means of computing, it does offer significant performance and operational benefits in various distributed applications. Tasks that were previously undertaken by client-server model have now been realized using mobile agents with performance benefits as well as ease of use and greater flexibility of operation. Mobile agents can behave autonomously and communicate with other mobile agents as well as their hosts while requiring minimal monitoring after their initial execution. Agents are powered with some level of intelligence, which permits development of powerful distributed applications.

Process calculi (also called *process algebra*) are mathematical tools used as formal methods for describing concurrent and communicating processes such as mobile agents. Although originally they have been used only for systems with static communication structure and static processes, process calculi have been adapted to model processes that change their locations as they progress such as mobile agents. APi-calculus [16], described later in this article, is one of the lastest and most comprehensive of such calculi and is a great asset for the mobile agent technology. Process calculi can be utilized to design complex systems involving mobile agents or processes in general, validate whether the resulting system performs according to the specification with which it was developed and analyze the performance of the system as well as compare it to similar systems that may be described with the calculus.

Although the benefits of APi-calculus cannot be denied in the field of mobile agents, there are some points that should be considered. It is a mathematical discipline that can appear very complex to a novice user and a system designer who is not well versed with such mathematics. It lacks the ability to describe to the user, at a glance, the structure and resulting behavior of a system. A visualization tool for such a calculus would be highly beneficial as it would allow the user to perform an initial evaluation of the system without having to go through the complex mathematical syntax and getting involved directly with the mathematical aspect.

There exist very few other practical tools for process calculi in general. Most of them have been developed for calculi with no support for systems with evolving communication structure, i.e., systems with inherent mobility [10]. For a list of such tools refer to [6], which presents some verification tools for process calculi with no support for mobility. Only in the last decade works have been done on verification tools for calculi such as Pi-calculus, which support evolving systems. Mobility Workbench (MWB) written in Standard ML [11] can be used for constructing and basic analysis of systems in Pi-calculus [22]. Extensions of Mobility Workbench have also been proposed, including one supporting polyadic Pi-calculus [21] and another for stochastic Pi-calculus [15]. Moreover, some high level programming languages have been developed that support Pi-calculus or its variants. These programming languages have a high level programming interface while the core of the system is based on a particular calculus. The most popular ones among these languages are Pict [13] and Nomadic Pict [23]. Others include TyCo[20] and HACL[5].

Saying that, the only existing *visualization* application for a process calculi is AmbIcobjs, developed by Damien Pous for the MIMOSA project in the summer of 2002 [14]. Ambicobjs is a visualization software implemented to work with Ambient Calculus (discussed in Section 2). Ambient Calculus is a more abstract calculus compare to the Pi-calculus family, with simpler syntax and structure, and therefore the AmbIcobjs development does not benefit the Pi-calculus and its variants.

In the rest of this article, after a brief introduction to process calculi and APi-calculus, the benefits and uses of such a visualization tool are discussed. Then the implementation details of the tool are presented. The article is concluded with a future work and a summary section.

2. A brief history of process calculi

Traditionally process calculi were used to model processes with static communication structure, therefore, they offered little support for mobility, which is based on a dynamic communication framework. Examples of such calculi include CCS [8] and CSP [4]. Tradition process calculi have been employed in various areas. Any system that can be reduced to a system of concurrent and communicating processes can be modeled and analyzed using such process calculi. Such systems include distributed applications, multithreaded processes and biological and chemical systems.

However, with the advent of mobile agents a need was felt for a process calculus that would effectively model such a system. The new process calculi support mobile agents as well as stationary agent systems. The Pi-calculus was the first, and along with its numerous variants, is still the major process calculus that supports mobile agents. Many other calculi have been introduced with varying degrees of popularity, including Ambient Calculus [14], a very simple yet expressive calculus. Following are some examples of process calculi that are popular in the mobile agents industry.

2.1. Pi-calculus

Pi-calculus has been the front-runner in the field of process calculi for mobile agents modeling. It was introduced in 1989 by Robin Milner et al. in a well-known paper [12] and was quickly recognized as an effective formal modeling tool for mobile processes. Pi-calculus was itself based upon a mobile extension of a process calculus called CCS, proposed by Mogens Nielson and Uffe Engberg [3].

Pi-calculus specifies two entities, names and agents or processes. A name can be a channel for communication as well as any data meant for communication by the agents. The communication by the agents is represented by a prefix; a negative prefix denotes an output of a name from the agent over a specified channel, again a name, while a positive prefix describes the agent receiving the name over the specified channel. For example, the expression,

$$\nu x\ (\bar{x}\, i.P \mid x\ (i).Q)$$

describes an interaction between two processes, or agents, *P* and *Q*. The separator '|' stands for concurrency and processes or agents existing on either side of it can exist concurrently and communicate with each other. The first part describes the fact that the name *i* is being passed by an agent *P* over the name *x,* which obviously serves as the link between the two agents. The second part describes agent *Q* receiving the name *i* over the channel *x. νx* denotes restriction of the name *x* to both agents *P* and *Q.*

The calculus provides a set of four *reduction rules* that describe how processes in their tasks. The *congruence rules* are used to decide if two syntactically different expressions are behaviorally equivalent.

Pi-calculus works with synchronous messages and is non-deterministic in nature. It stays away from higher order constructs although a Higher Order variant has been specified [19]. Rather it has been shown over the years through several researches that a higher order extension would not add to the expressiveness of Pi-calculus.

2.2. Higher Order Pi-calculus

Another way of modeling mobile communications is by passing actual agents or processes through a channel instead of passing just names. Such an extension of Pi-calculus, which allows agents to be passed as communication values on channels instead of names as in traditional Pi-calculus, called Higher Order Pi-calculus (introduced by Davide Sangiorgi in 1993 [19]). Due to the ability of passing processes or parameterized processes through communication channels, HOpi directly supports mobility.

However, as shown by Sangiorgi in the same paper such an extension to Pi-calculus is not of much practical use and does not add to the "expressiveness" of the calculus. Any expression that is expressed in HOpi can be expressed in Pi-calculus. Instead HOpi only serves to bring more complication and was introduced mainly to show that there is not a necessity of such higher order extensions.

2.3. Polyadic Pi-calculus

In contrast to Pi-calculus, which is essentially *monadic,* polyadic Pi-calculus proposed by Milner in 1991 [9] as an extension and refinement of the Pi-calculus, accepts *tuples* of names to be passed for communication. It does not add any expressiveness to the Pi-calculus but provides support for abstraction which itself gives more functionality to the calculus, and the ability to define several sorts over names. The notion of sorting plays the same role as the concept of typing in sequential programming and computations. It allows declaration of a name as a certain type or sort.

2.4. Ambient Calculus

A process calculus, quite different form Pi-calculus, for modeling mobile agent systems was proposed by Luca Cardelli and Andrew D. Gordon in 1998 [1], named Ambient Calculus. This is a very simple yet elegant and expressive calculus, and the central concept is that of an *ambient*, which is regarded as a closed place in which processes exist and perform their computation. An ambient can also have sub-ambients, and so on, each with its own set of processes and threads. The calculus is based on a few basic ambient primitives, such as *in*, *out*, *open* and *copy* that are sufficient to model mobile agents.

Process calculi provide an effective way of modeling systems and have been more popular than some other tools, such as Petri nets, for modeling concurrent processes. The process calculi described in this section are just a small section of such algebras available today. Most of these calculi are variants of Pi-calculus that have been developed for specific situations and provide extensions to support certain capabilities.

APi-calculus, described in the next section, was developed primarily to support intelligence in agents [17]. Other calculi provide no direct support for modeling intelligence in a system which is viewed as a disadvantage for designing powerful distributed applications, that include intelligence in the overall architecture. As detailed in the next section, besides intelligence, APi-calculus also addresses features such as security of the overall system and natural grouping of agents and hosts.

2. APi-calculus

APi-calculus was proposed as an extension to Higher Order Pi-calculus in 2002 by Rahimi [16] to provide features that were yet unavailable in other process calculi. Being an extension of the Higher Order Pi-calculus, it accepts processes to be passed over communication links unlike Pi-calculus, which only allows names to be passed through communication channels.

APi-calculus' main concern is to acknowledge intelligence while modeling a framework of mobile agents. Many mobile agent applications require a level of intelligence to be inherent in the mobile agents. That is quite easy to comprehend since the mobile agents are essentially self existing processes that are expected to move around over a network and communicate with each other, while performing assigned tasks. Agents may be required to cooperate with one another for efficient knowledge gathering and task performance. Many such tasks require some level of intelligence relative to the task complexity. Rahimi [14] gives a more detailed description of the characteristics of intelligent agents, which include: *Communication Ability*, *Capacity for Cooperation*, *Capacity for Reasoning and Learning*, *Adaptive Behavior* and *Trustworthiness.*

Another major contribution of APi-calculus is the concept of a *milieu*. With the introduction of the milieu to the calculus dictionary, natural grouping of mobile processes is addressed. This is a very powerful concept with a parallel to the concept of an ambient in Ambient Calculus. It provides the ability of grouping together hosts and processes that share common characteristics. This is an important factor when designing systems.

A major concern when dealing with mobile agent applications is the security of the system in which the agents operate. No user would like to allow any malicious code on his/her computer that would jeopardize its functioning and security. One of the reasons the mobile agent paradigm has not done that well is the distrust people associate with autonomous agents executing on their computers. APi-calculus does not at the moment directly define any support for security of mobile agent systems, but the current language can be used to address this issue. For example, the concept of milieu allows trusted hosts and agents to be grouped together. Any foreign agent that wishes to enter such a milieu could only enter if it shows proof that it could be trusted. Such proof could include issue of a key to the agent from a member of the milieu. The agent would be refused entrance in the absence of the key.

3.1. APi-calculus syntax

The APi-calculus syntax is similar to that of Higher Order Pi-calculus on which it is based. It adds support for terms, milieus and knowledge units. A brief overview of the syntax that is used to represent APi-calculus expressions is presented [18].

Terms. APi-calculus introduces the concept of *terms.* A term can be a name, a function, a fact or a rule, whereas in Pi-calculus the only term recognized is a name. A name can be a channel or a variable.

Term, T ≡

x, y, z	(name)
a, b, c	(fact / rule)
$f(x,y)$	(function)

A tuple of terms is represented as $\vec{x}$, where $\vec{x}$ represents x_1, x_2, x_3...

Processes. APi-calculus accepts processes to be sent over names. The following gives us a list of process expressions that are allowed in the calculus. α, representing action, can be either input or output action.

Process, P ≡

0	(empty)
$\alpha.P$	(action prefix)
$P_1 + P_2$	(summation)
$[T = R]\ P_1 : P_2$	(conditional)
$\nu x.P$	(name restriction)
$(K_i).P$	(knowledge restriction)
$!P$	(replication)
$D\langle\vec{L}\rangle$	(constant)

Knowledge Units. The knowledge unit is modeled to hold a set of rules and facts and is the basic unit upon which the intelligence of the system can be based. Information in a knowledge unit consists of facts and rules, which can be added or removed from it using the actions below. A knowledge unit can also be called using the parameters below.

Knowledge unit, K ≡

0	(empty)
r	(single rule in knowledge unit)
$K_1 + K_2$	(summation)
$(\vec{K}).P$	(knowledge name restriction)
$K_i(\vec{a})$	(add tuple $\vec{a}$ to K_i)
$\vec{K}_i\ a$	(drop a from K_i)
$K_i\langle\vec{a}\rangle(\vec{R})$	(knowledge unit call)

Milieu. As noted before a milieu is a new concept introduced in APi-calculus that provides for a natural grouping of processes and hosts and provides features for security. The two features that would support security are the join and leave functions that must be used for an agent to enter or leave the environment of a milieu.
Following is a list of milieu related expressions that are used in APi-calculus.

Milieu, M ≡

0	(empty)
$M[O_1 + O_2]$	(composition)
$M_1 + M_2$	(summation)
$join\ p.M$	(p joins milieu M)
$leave\ p.M$	(p leaves milieu M)
$open\ M$	(opens M's boundary)

Free and bound term. Free and bound terms are very important terms that are central to how a process communicates and progresses. A bound term is a term that is restricted or private to a process, i.e., it is local to the process and it cannot be used by that process as a channel to communicate with other processes. Any term in a process that is not bound to it is called a free term and can be used for communication. The free terms of a process P are represented as $ft(P)$.

Reduction. APi-calculus defines certain inference rules, which govern reduction. Reduction can be used to simplify calculus expressions and is represented by the symbol '$\rightarrow$'. It comes about by changes in a process and a process is said to 'evolve'. A set of axioms for *structural congruence* are also defined. Structural congruence is an important tool when manipulating expressions and equations.

For example, $O \rightarrow O'$ implies that the process or milieu O evolves to O' due to certain actions inside O. The concept of structural congruence, which is represented by $\equiv$, plays an important role in such reductions, as mentioned before.

For example, if

$$O_1 \equiv O_2 \text{ and } O'_1 \equiv O'_2, \text{ and } O_2 \rightarrow O'_2$$

then the following can be inferred,

$$O_1 \rightarrow O'_1$$

The reduction rules are central to the analysis component of the visualization tool discussed in this article. The progress of a system is understood by applying the reduction rules, which is then visualized on the screen. Details regarding the implementation of the analysis tool are presented in Section 5.

4. The Need for a Visualization Tool for APi-calculus

The reduction rules described in the last section form the core of expressing the evolving nature of a system described in APi-calculus. However, performing these reductions manually on a calculus expression can be cumbersome and error-prone, especially for systems with complicated communication structure. A tool for performing automated reduction on calculus expressions, therefore, becomes very important when considering the practical significance of the calculus. The software presented in this paper includes such a tool. Along with performing the reduction of the calculus expressions, the software performs the visualization of these expressions, which gives a visual understanding of the system's progress. In this sense, the program behaves as a "visual compiler".

It is a widely accepted fact that visual descriptions make a much better impact than a textual representation [2], although they may not be as descriptive. Certain concepts are much easier to discern and understand visually than in any other way [7]. Visual aids are especially important and almost necessary where there is a need of describing relationships between entities that may not be evident in a textual description. Again, a textual description for a dynamic system, along with explanation of the progress over time, can never convey the full meaning of the dynamic nature of the system. A visual description of the same would be more explanatory and helpful. In the least, a textual or mathematical description can be aided a great deal by using a visualization tool.

The major intended function of APi-calculus is to describe mobile agent systems, which typically demands a very dynamic environment. Mobile agents travel from host to host, communicate through various channels and engage in several execution cycles and message passing throughout their life cycle. What APi-calculus does, is describe these interactions and migrations in a definite, crisp and abstract manner with the help of mathematical semantics.

The visualization tool for APi-calculus, named ACVisualizer (*A*Pi-calculus *V*isualizer) serves as a rudimentary visual compiler for the APi-calculus. In the sense of a traditional compiler, the ACVisualizer takes the raw APi-calculus syntax, analyzes the progress of the calculus expressions with the help of calculus reduction rules, and then outputs this progress for the user to observe. The next section discusses how this progress can be quantified to further benefit the designer and developers by providing a solid analytical tool.

4.1 Benefits of a Visualization Tool

Although no other visual tools exist for any process calculi related to Pi-calculus, the benefits of such a tool can be manifolds. Following is a list of the benefits of a visualization tool for APi-calculus.

Enhancing the APi-calculus as a companion application. ACVisualizer can be used along with any APi-calculus model in order for the user to better understand "what is going on" in the system. Mathematical syntax can appear to be highly abstract and even adding textual description would not be able to provide the user with a higher-perspective understanding of the system. For example, a visualization tool can be used to observe relationships that persist over a period of time, which may not be very obvious from a mathematical description. Using mathematical syntax, progress is usually depicted step by step and the significance of a relationship between components of a system may not be very apparent to the user.

After writing a preliminary description of a system in APi-calculus syntax, the user may then use the ACVisualizer to assess how the system behaves in quick snapshots. This can also help the user to assess another designer's description of a system without going through the trouble of consuming the mathematical syntax and drawing inferences from them.

Acting as a "debugging tool" for the designer. When used in conjunction with APi-calculus, ACVisualizer gives the designer a quick look at the elements that he/she has wrongly supposed. The user may go through the whole system life cycle and progress step by step to analyze the problems the current design may be facing. These problems may fall under security or performance issues. Security issues may include a mobile agent unwittingly joining a particular milieu that it should abstain from accessing certain resources it does not have permission for. Security has in fact been one of the core reasons why the mobile agent technology has not been given its due till now and maybe rightfully so. Although APi-calculus has immediate means for expansion to include security issues, it currently does not address this element of mobile agents framework directly. A visual tool would help greatly here by providing clues to an observant user.

A user may also be looking out for performance concerns in the system that may not be apparent when describing it in the calculus. A mobile agent may have a more straightforward route for its itinerary than the one it is instructed to take but this might be hidden in the calculus syntax and may not be obvious. In another situation, a shared resource may not be optimally accessed by the agents interested which may lead to a performance hit. All this would be evident when the system is visualized and the problem can be visually identified. A timer would serve as an actual measurement in such cases and can be used to quantify the problems formally. Saying this, ACVisualizer is not meant to replace formal calculus based methodologies such as bisimilarity, for in depth debugging of the system models.

A comparison tool. Another application of the ACVisualizer is to serve as a comparison tool for system design and analysis. Different designers may come up with differing implementations for the same system; ACVisualizer could help in initial determination of which implementation might be more effective. This can be beneficial, for instance, when a mobile agent framework needs to be compared to a client-server framework designed to fulfill the same task, and the effectiveness of the mobile agent framework vs. the client-server could be evaluated.

Applicable to other process calculi. There is currently a dearth of any kind of graphical tool for process calculi. As mentioned earlier, the only such tool available is AmbIcobjs, for Ambient Calculus, which has a different structure and semantic than Pi-calculus. On the other hand, APi-calculus is an extension of the Pi-calculus and the implementation of the ACVisualizer would aid other process calculi based on Pi-calculus as well. The ACVisualizer could be modified to support the other extensions with minor modifications to the implementation.

APi-calculus syntax conversion to machine-readable format. The new syntax introduced in the next section, mimicing the APi-calculus syntax, provides an easy medium in which the calculus expressions can be used in machine-readable format. This format could be used in any tool developed for Pi-calculus and its extensions.

5. The Implementation of ACVisualizer

ACVisualizer, implemented in Java, is capable of reading and analyzing APi-calculus and Pi-calculus expressions involving processes, milieus and knowledge units. The implementation can analyze and visualize systems described in PI or API calculus expressions. It must be noted that since Pi-calculus is considered a subset of APi-calculus, this section describes the implementation details using only APi-calculus syntax.

5.1 The Syntax

The implementation of the calculus required the specification of a new syntax, which mirrors that of the APi-calculus but provides certain constructs that are essential for the purpose of implementation. The new, plain text and more readable syntax does not add an extra level to the syntax or the semantics of the calculus; it is adopted mainly to allow an ease of specification of calculus expressions.

Terms do not undergo much change except for the representation of the tuple. In the examples in this paper, the knowledge of a term being a tuple of terms is mentioned in the meta data description that is read before reading the expressions. Therefore there is no distinction made in the representation of a or $\overline{a}$ in the ACVisualizer syntax. Tables 1, 2, 3, and 4 present ACVisualizer syntax for different expressions.

Table 1. Terms

Expression	**APi-calculus**	**ACVisualizer**
Name	X	X
Fact or Rule	A	A
Function	f(a, b, c)	f(a, b, c)

Table 2. Processes

Expression	**APi-calculus**	**ACVisualizer**
No Action	0	0
Input Prefix	x(L).P	input(x, L).P
Output Prefix	$\overline{x}$L.P	output(x, L).P
Parallel	$P_1 \mid P_2$	P1 | P2
Summation	$P_1 + P_2$	sum(P1 + P2)
Conditional	$[T=R]P_1{:}P_2$	if (T = R) then (P1, P2)
Name Restriction	ν xP	restrict(x, P)
Knowledge Name Restriction	(K)P	restrict(K, P)
Replacement	$\{L_1/L_2\}$	{L1/L2}
Replication	!P	!P

Table 3. Milieu

Expression	**APi-calculus**	**ACVisualizer**
Empty	0	0
Include	M[O1 | O2]	M(O1, O2)
Composition	$M_1 + M_2$	M1 + M2
Join	Join m.P	join(P, m)
Leave	leave m.P	leave(P, m)

Table 4. Knowledge units

Expression	**APi-calculus**	**ACVisualizer**
Empty	0	0
Single Rule	*R*	R
Summation	$K_1 + K_2$	K1 + K2
Restriction	$(\overline{K}).P$	restrict(K, P)
Drop Fact/Rule	$(\overline{K})a$	drop(a, K)
Add Fact/Rule	$K(\vec{a})$	add(a, K)
Knowledge Unit Call	$K<\vec{a}>(\vec{R})$	call(a, K, R)

The syntax represented in these tables is a simplified representation of APi-calculus syntax. Since this new syntax is more intuitive than the mathematical one, the calculus expressions could be exclusively written in the new syntax especially if they are meant to be primarily analyzed by ACVisualizer or other analysis tool.

Before discussing the ACVisualizer components, a review of two systems expressed in APi-calculus and their visualization is discussed below. These are example scenarios of the use of the visualization tool. The examples in this section are described in [16] in detail.

Example 1: Passing Knowledge Units between Processes. Suppose agent *P* wishes to pass knowledge unit *K* to process *Q* over channel *x*. This knowledge unit *K* could be a private name to *P,* or it may be non-private. Both cases are discussed below.

a) Non-private Knowledge Unit: In this case, *P* simply passes *K* to process *Q*, as in the expression:

$$\overline{x}K_1.P' \mid x(K_1').Q' \rightarrow P' \mid Q'\{K_1 / K_1'\}$$

In this example, P is $\bar{x}K_1.P'$ and Q is $x(K_1').Q'$. The transition is shown by the right hand side of $\rightarrow$, with K_1 replacing K_1'. In the new syntax this is represented as:

output(x, K_1).P' | input(x, K_1).Q' $\rightarrow$ P' | Q' {K_1 | K_1'}

The above transition is graphically shown in Fig. 1.

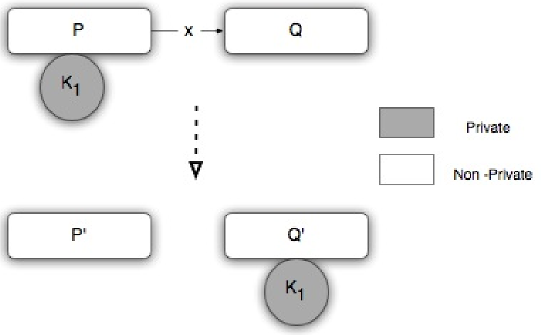


Fig. 1. Non-private knowledge unit passing

b) Private Knowledge Unit: When the knowledge unit *K* is private to the process *P,* it passes a copy to *Q* over *x*. This is shown by the expression.

$$(K_1)(\bar{x}K_1.P') \mid x(K_1').Q' \rightarrow (K_1)(P' \mid Q'\{K_1 / K_1'\})$$

In the new syntax this is written as:

restrict(K_1, output(x, K_1).P' | input(x, K_1').Q' $\rightarrow$ restrict(K_1, P' | Q' {K_1, K_1'}

Graphically, this can be shown as in Fig. 2.

Example 2: Processes and Milieus Joining Milieus. The following expression describes two processes, P_1 and P_2, joining milieu M_1, and then milieu M_1 joining milieu M_2.

$$join\ M_1.P_1 \mid join\ M_1.P_2 \mid M_1[0] \rightarrow M_1[P_1 \mid P_2]$$
$$join\ M_2[0].M_1[P_1 \mid P_2] \rightarrow M_2[M_1[P_1 \mid P_2]]$$

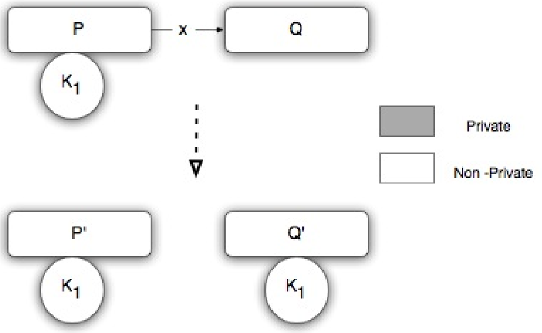


Fig. 2. Private knowledge unit passing

In the new syntax, the two expressions are written as below:

join(P_1, M_1) | *join*(P_2, M_1) | M_1(0) $\rightarrow$ M_1(P_1 | P_2)
join(M_1 (P_1, P_2), M_2(0)) $\rightarrow$ M_2(M_1(P_1, P_2))

The graphical representation of these transitions is illustrated in Fig. 3.

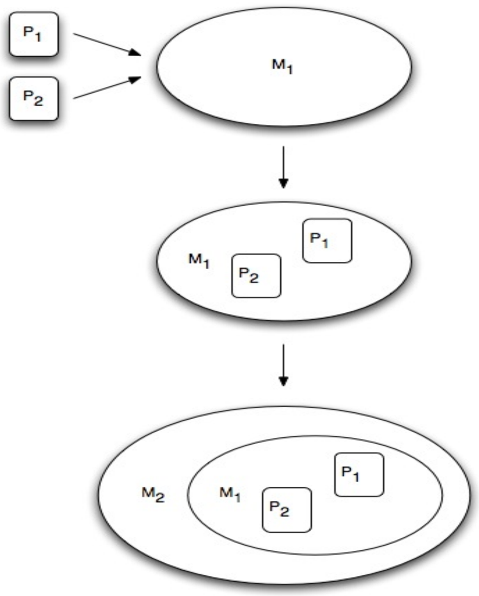


Fig. 3. Processes and Milieus Joining and Leaving Milieus

5.2. Components of ACVisualizer

Fig. 4 illustrates the process of visualization in general and the three modules that form the program. Separating the tasks of the program into individual modules in this manner allows for relatively easier extension of the system.

The Input Module. The input module is responsible for reading in the information about the system and storing it in an easy to consume format. This includes the calculus expressions as well as the meta-data describing the components involved in the expressions. The meta-data would include information regarding the variables used, special information regarding the system and comments about the expressions. The input file starts with the meta-data explaining the terms and processes used in the expressions in the file. The meta-data may include names of the processes, the bound and free names associated with each of them, the milieus and the knowledge units.

Parsing and Analysis Module. This module performs the task of parsing and analyzing an expression written in the calculus and forms the core of the visualization program. The system expression is broken down, after analysis, into a collection of objects that represents the structure and semantics of the original expression in terms of the object-oriented framework of the implementation. This module incorporates the core logic of the calculus in the program, including the reduction rules. In general, it acts as a compiler for the calculus, for the specific purpose of the visualization, and can be extended to support other calculi if needed.

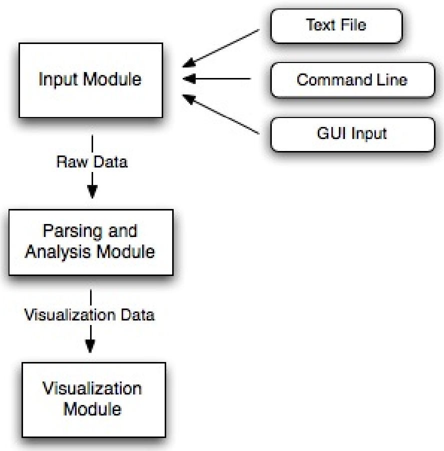


Fig. 4. ACVisualizer's modules

As explained earlier, any calculus expression is the formal representation of the initial state of a system, including the configuration of the system's components and their capacity for interaction at this stage. When the input module passes such an expression to the parsing and analysis module, it must be pre-processed for the visualization of the evolving nature of the system. This preprocessing can be understood as a three-step process as follows, with detailed explanation provided thereafter:

Analyzing the system expression for identification of the components of the represented system. These include all the primitives of the calculus: for e.g., for API calculus, the module should identify the elements such as processes, milieus, terms, etc. in the source expression. Along with these primitives, there are certain implementation-specific abstractions that are identified at this stage, which are crucial for the working of the program. For example, the implementation identifies a construct called *Verb* that is the extension of the basic *action* primitive of the calculus.

Forming the object-oriented model of the system. This is a crucial, yet a very flexible element of this module. Forming another representation of the system expressed in calculus requires an understanding of the original calculus as well as proper object-oriented design practices. The choices made regarding the class structure and composition affect the design of the visualization program and various issues including performance, ease of implementation, extensibility, and correctness of the overall model have to be borne in mind.

Analysis of the dynamic nature of the system. This is the most intensive component of the whole program. An expression in calculus represents the initial state of the system, and APi-calculus, as other calculi, defines *reduction rules* that are used to analyze the possible progress of the system from this initial state. This module is responsible for generating such a progress report that is eventually used by the visualization module, either on the fly as the progress is being analyzed, or after the complete generation and storage of the state changes of the system.

Whereas the first two above are handled by the parsing element of the module, the third task is performed by the analysis element. It must be noted that for implementation, there is not always a clear distinction between the two elements of the module.

The parsing element is responsible for processing the input expression in calculus syntax and producing the object-oriented (OO) model of the represented system. Although process calculus is traditionally perceived to be a procedural representative of a system with its emphasis on the reduction rules, the Pi-calculus family can be used to represent the overall structure of the system (with communication primitives). APi-calculus adds to the structural representation with the introduction of milieus. In light of this, an OO model of the calculus expressions, with its emphasis on the structural composition of a system, gives a conceptual clarity for understanding the represented system. As in any translation to an OO model, the eventual model may not be complete in all respects and may need to be refined in future iterations of the design. We consider our OO translation strategy to be very effective, which includes most of the primitives and rules of APi-calculus and hence Pi-calculus.

The implementation recognizes the following constructs in the calculus expressions, which map to the corresponding classes in the OO model:

Names and Terms: they correspond to those in the APi-calculus syntax and are recognized as such.

Verbs: a verb is derived from a process and its set of actions that exist simultaneously with other components of the system. A verb, in most cases, is a process that is separated syntactically from other processes by a '|' in the calculus. Therefore it includes the suffix process, which could be null, and the prefix actions. Each verb also has a condition associated with it that provides the functionality for match and mismatch. This field has the form sign|var1,var2. Sign is one of 3 numbers 0 indicating mismatch, 1 indicating match, or 3 indicating no condition. The two variables are the variables considered in the condition. So the match [x=y] would be represented by 1x,y and the mismatch [x!=y] would be represented 0x,y. Whenever no condition is present 3 is used.

Processes: they correspond to the process in the calculus syntax for the most part. The corresponding object for the process would hold extra information for proper modeling:

a. The name of the process
b. The verb it may belong to
c. The immediate milieu that the process belongs to, if any

Milieus: a milieu is recognized as such along with any knowledgebase, process, or milieu that it may contain.

Knowledgebase: they are recognized along with any process that they may belong to. Although the idea of knowledgebase has not been expanded yet to realize their power in intelligent-systems design, eventually this class would be expanded to hold specific information regarding the information that a knowledgebase may hold.

Line progress: a line progress holds the state of the system at a particular instance. The implementation subsection provides more details.

The analysis element of this module is implemented by an expert system that can perform the reduction rules, prescribed by the calculus, on a given expression. These reduction rules, as explained earlier, are used to define the progress of a system. Implementing a reduction system like this in a traditional programming environment, without the aid of an expert system, can be complex and inefficient. Expert systems are convenient for specification of a set of rules that can be used to define a decision system with ease and flexibility. They employ pattern-matching techniques, such as the Rete algorithm, efficiently.

Visualization Module. As the name suggests, this module is responsible for the final graphical representation of the system originally described by the calculus expression as well as its progress. The parsing and analysis module described in the previous subsection provides the visualization module with the necessary information, in the form of a line progress deduced by the reduction rules, that is used to represent the system with its component. The OO-model constructed by the previous module is valuable for visualization purpose, since it defines objects that map directly to the visual representation of the components of the system. The progress of the system from one reduction to the next can be represented and the status area is updated with every step of the reduction regarding the evolution of the system.

This module also provides the user with an interface to interact with the program. The visualization interface is informative, easy to use and provides means for further enquiry by the user. For instance, the "update section" of the interface allows the user to modify components and parameters of the input model and observe the changes by refreshing the visualized system (Fig. 7).

To illustrate the progress of the system, from the initial state to the last state, the visualization module provides the user with different options. The user may choose to look at the progress of the system from the beginning to the end, or she may decide to interact with the software and look at the individual states, start, freeze and stop the graphics. Status boxes to the right side of the screen inform the user about the status of the system progress.

5.3 The Structure of the Implementation

The major implementation of ACVisualizer is done in Java. The structure of the software is designed to ensure enough modularity to support addition of more features and integration of external components. Fig. 6 illustrates the more important Java classes in a UML class diagram, while Fig. 5 shows the UML package diagram for the Java implementation with details regarding the functionalities.

The following are the major classes and their package qualifiers used in the implementation:

- *pivisual.display.GUI*: It takes care of the user interfaces and visualization components of the system.
- *pivisual.engine.parsing.Parser*: It is responsible for parsing the meta-data as well as defining the system, given as input, in the form of Java objects.
- *pivisual.engine.parsing.MetaInfo*: The Parser stores the parsed information concerning the meta-data in an object of this class.
- *pivisual.engine.parsing.Verb*: This is the smallest unit of action that is comprehended by ACVisualizer and can be visualized by the *GUI* object. As shown in the UML class diagrams, the *Verb* class is inherited by several other classes that implement the actions
- *pivisual.engine.parsing.LineProgress*: a collection of *Verb*s make a *LineProgress*, which represent a single line that is written in the calculus.

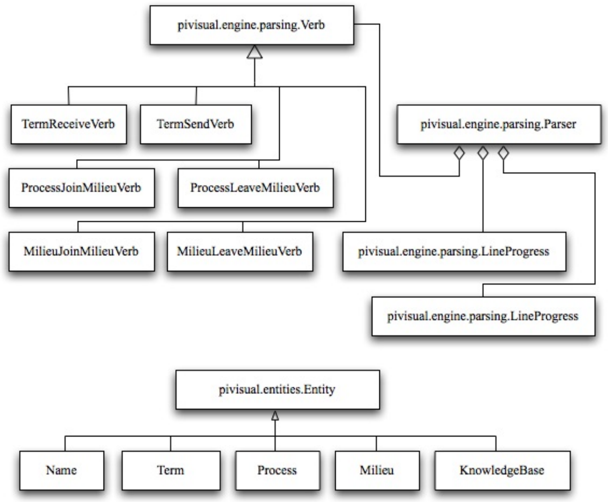


Fig. 5. UML class diagram

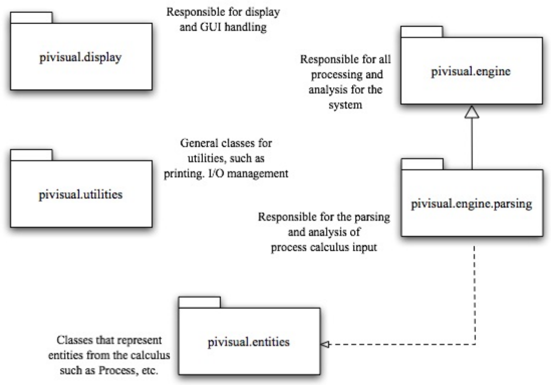

Responsible for display and GUI handling
pivisual.display
Responsible for all processing and analysis for the system
pivisual.engine
General classes for utilities, such as printing. I/O management
pivisual.utilities
pivisual.engine.parsing
Responsible for the parsing and analysis of process calculus input
Classes that represent entities from the calculus such as Process, etc.
pivisual.entities

Fig. 6. UML Package diagram

The following classes represent the various entities that can be represented in the calculus:

- *pivisual.entities.Name*
- *pivisual.entities.Term*

- *pivisual.entities.Process*
- *pivisual.entities.Milieu*
- *pivisual.entities.KnowledgeBase*

The above described classes form the core object-oriented model of the calculus. Along with this the analysis of a system described in calculus is performed by the expert system developed to facilitate the application of reduction rules. The expert system is developed in the Java Expert System Shell (Jess) [24], a version of the CLIPS expert system framework [25] ported to Java. Interfacing Jess with the parsing and visualization modules becomes simpler because of similar platforms. The developed expert system employs Rete algorithm for effective pattern matching. The patter matching technique is very effective in analyzing calculus expressions for determining their progress.

5.4 A Simple Visualization Example

In this section, a visualization example of an APi-calculus model, described in ACVisualizer syntax, is presented. The example consists of four processes (P, Q, R and S), three milieus (M, N and O), and two knowledge bases (K1 and K2). The evolution of the system includes processes sending and receiving various names and knowledge units and leaving and joining the milieus. The content of the input file to ACVisualizer is shown below and should be self-explanatory.

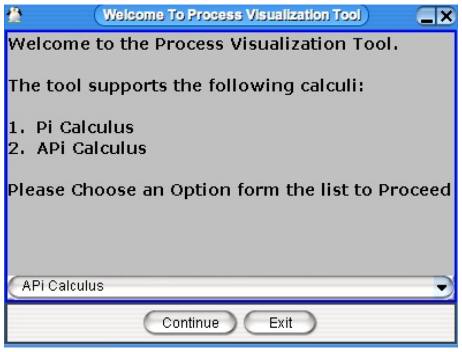
```
//PR P Q R S
//ML M N O
//KB K1 K2
//CONTAINP P M
//CONTAINK K2 M
output(x, L).P|input(x, L).Q
join(S, N)
output(y, J).P|input(y, J).S
output(y, J).R|input(y, J).Q
join(R, N)
join(K1, N)
leave(K1, N)
```

The first 5 lines form the meta-data of the model that describe the elements of the system being modeled and their initial states. The last 8 lines describe the progress of the system step by step and include all the dynamic entities of the model described in the previous paragraph. When the visualization program is started with an input file like above, the program first initializes the main screen according to the meta-data. After the initialization the progress of the system is either automatically visualized or it is dictated by the user interaction through the use of the "Next" and "Previous" buttons at the bottom right of the window. The status of the system is shown in two status boxes on the right side of the screen. Fig. 7 demonstrates two views of ACVisualizer. The one on the left shows the startup window, which lets the user choose from APi-calculus and Pi-calculus as the language for specification. The second window shows the progress in one of the intermediate steps.

5.5 A Mobile Communication System Example

We now present a more complex and practical example of formal modeling and visualization of a mobile communication system. The system consists of a mobile phone user traveling in a car with the phone in direct communication with different transmitters, representing different signal areas at different times. The transmission towers themselves are in communication with a control center that is responsible for managing the whole system.

In this example we consider a central control center, *ControlCenter*, two signal regions, *Region1 and Region2,* each with a transmitter, *Trans1* and *Trans2*, and a single car, *Car*. The following actions show certain progress in the system's communication structure: *Car* talks to *Trans1*; *Trans1* is instructed by *ControlCenter* to allow the signal

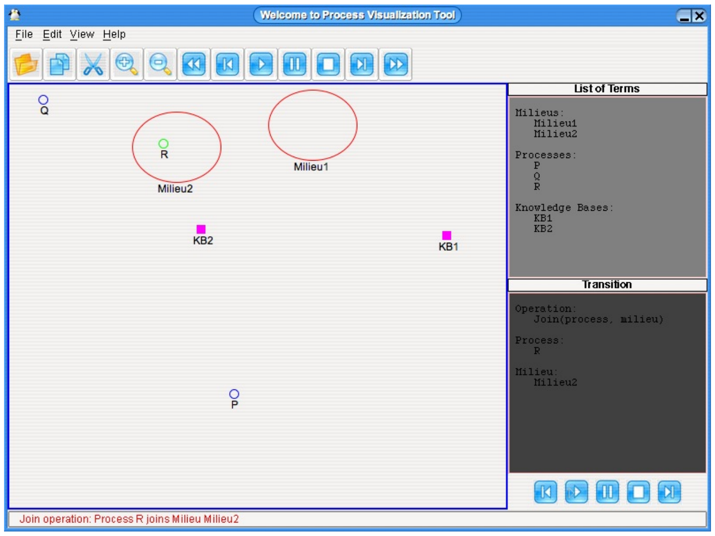

Welcome To Process Visualization Tool

Welcome to the Process Visualization Tool.

The tool supports the following calculi:

1. Pi Calculus
2. APi Calculus

Please Choose an Option form the list to Proceed



Milieus:
  Milieu1
  Milieu2

Processes:
  P
  Q
  R

Knowledge Bases:
  KB1
  KB2

Operation:
  Join(process, milieu)

Process:
  R

Milieu:
  Milieu2

Join operation: Process R joins Milieu Milieu2

Fig. 7. ACVisualizer in action

for *Car* to switch over to *Trans2*; *Trans1* instructs *Car* to switch over to *Trans2*; *Car* switches over to *Region2* and talks once to *Trans2*. Obviously, *Car*, *Trans1*, *Trans2*, and *ControlCenter* are modeled in APi-calculus as processes, while *Region1* and *Region2* are modeled as milieus. The following single expression encapsulates and expresses the above system progress:

$$Region1[\overline{talk1}.switch(r,t,s).leave.join r.\bar{t}.Car \mid$$
$$talk1.l.lose(r',t',s').j.\overline{switch}(r',t',s').Trans1]$$
$$\mid Region2[talk2.Trans2] \mid$$
$$lose(Region2,talk2,switch2).ControlCenter$$

The *talk* channel represents communication between the car and the transmitter, the *switch* signal is sent by the transmitter to the car with the following parameters: the region to which *Car* should switch, and the *talk* and *switch* channels of the new region's transmitter. Leave and join are normal milieu related operations, and are abbreviated as *l* and *j* when these operations are purely for the sake of communication with an outside-milieu process.

The above expression can be further analyzed through reduction to see if it does perform according to the desired specifications. The first reduction would be:

$$Region1[switch(r,t,s).leave.join r.\bar{t}.Car \mid$$
$$l.lose(r',t',s').j.\overline{switch}(r',t',s').Trans1] \mid$$
$$Region2[talk2.Trans2] \mid$$
$$lose(Region2,talk2,switch2).ControlCenter$$

where the *talk1* channel is used for communication. The next reduction is:

$$Region1[switch(r,t,s).leave.join r.\bar{t}.Car \mid$$
$$\overline{switch}(r',t',s').Trans1] \mid$$
$$Region2[talk2.Trans2] \mid ControlCenter$$

Here *Trans1* has to leave *Region1*, receive the lose signal from the *ControlCenter*, and join back. Notice the parameters of the lose signal: the region to which *Car* has to switch, the channel it should communicate on in the new region, and the channel for the next switch signal. If there were more than one car, the lose signal could be parametrized with a parameter for the particular car. Next reduction:

$$Region1[leave.join Region2.\overline{talk2}.Car \mid Trans1] \mid$$
$$Region2[talk2.Trans2] \mid ControlCenter$$

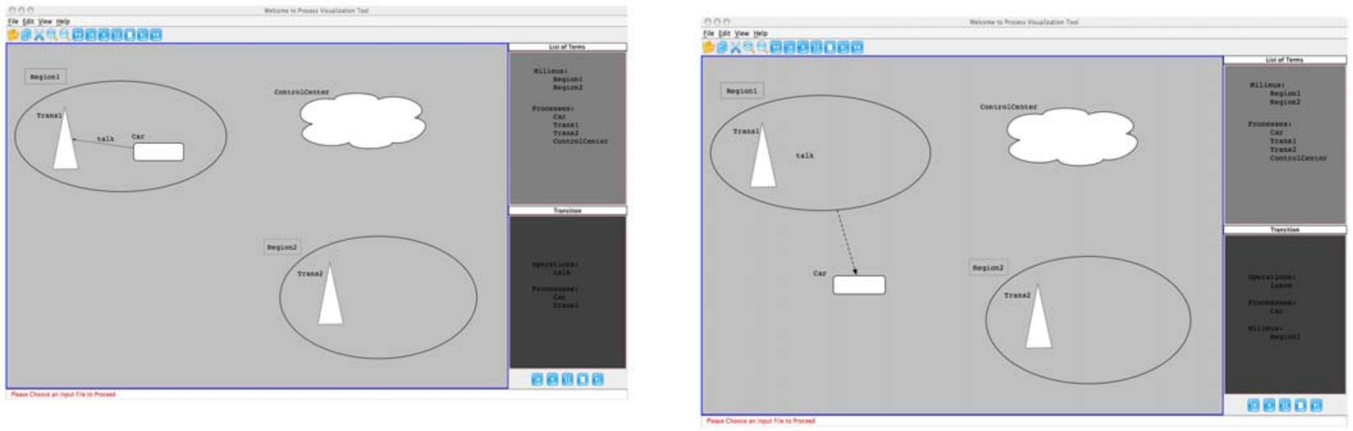


Fig. 8. Mobile Communication visualized in ACVisualizer

where *Car* receives the switch signal from *Trans1*. Notice the substitutions in *Car*'s prefix. The final set of reductions would result in:

$$Region1[Trans1] \mid Region2[Car \mid Trans2] \mid ControlCenter$$

where *Car* leaves *Region1*, joins *Region2*, and then talks with *Trans2*.

Although the above system has been emulated quite thoroughly by the calculus, it would be very helpful to peruse its dynamic nature visually. Specifically the reductions that we performed manually would be automated, and then demonstrated. The graphical representation of the states of the system after two of the above reductions is shown in Fig. 8. The software is capable of animating the progress, while providing the user with the interface to step through the individual states.

6. Future Work and Conclusions

6.1. Future work

This work is the first step in the development of a complete tool that can be used for design and analysis of complex systems such as multi agent architectures. This section provides some intended future works under this topic.

The current software implements the majority of the syntax and semantics of APi-calculus, in terms of compiling/visualizing the calculus expressions. Furthermore, adjusting the software to support other process calculi in Pi-calculus family, such as higher order Pi-calculus, synchronous Pi-calculus, Spi, etc., is undemanding and involves some basic changes to how the input is read and analyzed in the program.

Although ACVisualizer, as a visualization tool, is helpful for system analysts and designers by itself, integration of other functionalities such as a system verification mechanism into ACVisualizer could be very beneficial. The verification process provides a means by which the functionality of a system can be verified according to the desired properties. Verification involves using mathematical techniques developed for API and PI calculi such as bisimilarity to check whether a system is behaving as it should. A verification tool can provide results according to certain parameters regarding the functionality of the system. Results provided by such a verification mechanism can be then visualized by the visualization application. This visualization can provide visual clues that can aid in analyzing the results of the verification.

As depicted in Fig. 9, the visualization process can take place without verification; however, in case verification is desired in the analysis, it would appear as an intermediate step. The visualization in this case would be more informative to help the user with the possible real-time modifications to the system.

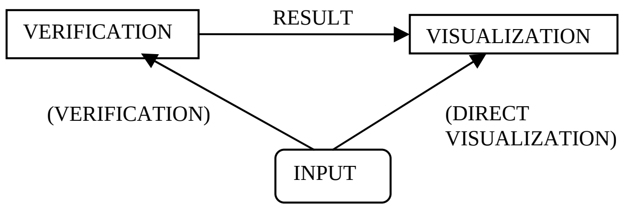


Fig. 9. Verification tool with visualization

Another important area of development that would be targeted next is adding support for quantifying system performance. API and Pi-calculus do not include mathematical facilities for performance evaluation. Currently the authors are in the process of developing theoretical infrastructures to be used for performance evaluation of the systems implemented in the calculi.

6.2. Conclusion

Mobile agent is a growing technology that is rapidly finding support in various areas and has been utilized widely. It is a relatively fresh technology and its applications demand an in depth understanding due to its complexity and unpredictable nature. Process calculi such as APi-calculus serve as tools that aid in the specification and analysis of mobile agents and more generally multi-process systems. In order to fully utilize these tools, a program to visualize systems described in mathematical terms is highly beneficial. It can be used to provide a wide perspective understanding of the system and to draw quick inferences about its the structure and performance. The syntax proposed in this paper for transforming APi-calculus expressions has its own merits and can be useful in making APi-calculus, and in general Pi-calculus more accessible.

ACVisualizer does have its shortcomings. It may not be reliable enough for a thorough evaluation of models since visual clues might slip the attention, especially in a complex system with constantly evolving structure. The current version of the software, by itself, does not provide any formal tools for validation, analysis and performance evaluation. However, progress is being made on this front in terms of the theoretical details as well as the implementation.

In conclusion, the visualization tool proposed in this paper in conjunction with APi-calculus can be highly effective in the description and analysis of intelligent mobile agent systems as well as other distributed multi-process systems.

References

[1] L. Cardelli and A. D. Gordon. Mobile Ambients. Foundations of Software Science and Computational Structures, Maurice Nivat (Ed), LNCS 1378, Springer, pp. 140-155, 1988.
[2] C. Diezmann. Effective Problem Solving: A Study of the Importance of Visual Representation and Visual Learning, Seventh International Conference on Thinking, Singapore, 1997.
[3] U. Engberg, and M. Nielson. A calculus of Communicating Systems with label–passing, Report DAIMI PB 208, Computer Science Department, University of Aarhus, 1986.
[4] C. Hoare. Communicating Sequential Processes. Prentice Hall, 1985.
[5] N. Kobayashi. Concurrent Linear Programming. PhD Thesis. Department of Computer Science, University of Tokyo, April 1996.
[6] E. Madelaine. Verification tools from the CONCUR project. Bulletin of the European Association of Theoretical Computer Science 47, pp110 – 126, 1992.
[7] C. McLoughlin and K. Krakowsi. Technological Tools for Visual Thinking: What does the research tell us?, AUC Academics & Developers Conference, 2001.
[8] R. Milner. Communications and Concurrency, Prentice Hall, 1985.
[9] R. Milner. Polyadic Pi-calculus: a Tutorial. Technical Report ECS-LFCS-91-180, Laboratory for Foundations of Computer Science, Department of Computer Science, University of Edinburg, UK. October 1991.
[10] R. Milner. Communication and Concurrency. Prentice Hall. 1989.
[11] R. Milner. M. Tofte and R. Harper. The definition of Standard ML. MIT Press, 1988.
[12] R. Milner. J.G. Parrow, and D.J. Walker, A Calculus of Mobile Processes. Part I and II, tech report ECS-LFCS-89-86, Edinburgh University, 1989.
[13] B. C. Pierce and D. N. Turner. Pict: A Programming Language based on the Pi-calculus. Technical Report CSCI 476, Computer Science Department, Indiana University, 1997.
[14] D. Pous. Les Mobile Ambients en Icobj. Internship report for the MIMOSA (Migration

and Mobility: Semantics and Applications) project, Summer 2002.

[15] C. Priami. Stochastic-Calculus. The Computer Journal, 38(7), pp. 578--589.

[16] S. Rahimi. APi-calculus for Intelligent-Agent Formal Modeling and its Application in Distributed Geospatial Data Conflation, PhD Dissertation, Department of Computer Science, University of Southern Mississippi, 2002.

[17] S. Rahimi, M. Cobb, D. Ali, and H. Yiang. An Intelligent representation in agent systems: an extended Pi-calculus. Proceedings of the 2002 IEEE International Conference on Fuzzy Systems, 2002.

[18] S. Rahimi, M. Cobb, D. Ali, and F. Petry. A Modeling Tool for Intelligent-Agent Based Systems: APi-calculus. Soft Computing Agents: A New Perspective for Dynamic Systems, The International Series "Frontiers in Artificial Intelligence and Application" by IOS Press, pp. 165-186, 2002.

[19] D. Sangiorgi, From Pi-calculus to Higher-Order Pi-calculus - and Back. TAPSOFT, 1993.

[20] V. T. Vasconcelos and K. Honda. Typed Concurrent Objects. In Proceedings of the Eighth European Conference on Object Oriented Programming (ECOOP), volume 821 of Lecture Notes in Computer Science, pp. 100 -117. Springer-Verlag, July 1994.

[21] B. Victor. A Verification Tool for the Polyadic Pi-calculus. Uppsala Thesis, Sweden. 1994.

[22] B. Victor and F. Moller. The Mobility Workbench – a tool for the Pi-calculus. In D. Dill, Editor, Proceedings of CAV'94, Lecture notes in Computer Science. Springer-Verlag, 1994.

[23] P. T. Wojciechowski and P. Sewell. Nomadic Pict: Language and Infrastructure Design for Mobile Agents. IEEE Concurrency, 8(2): pp. 42 – 52, 2000.

[24] Ernest Friedman-Hill, Jess In Action: Java Rule-based Systems, Manning Publications Co., 2003

[25] J. Giarratano, G. Riley, Expert Systems: Principles and Programming 3rd eds., PWS Publishing Company, 1998