Formal Approach to an Architecture of Distributed Information Systems

Marek Rychlý*

*Faculty of Information Technology, Brno University of Technology
rychly@fit.vutbr.cz

Abstract

An architectural design of a distributed information system and its implementation can be very complex and difficult – especially in a case of dynamic architecture, which is changing in a system runtime according to behaviour of the system. Formal approaches can eliminate most failures by model checking, but they require a formal specification of architecture in the design phase of a project. In many cases, this strong dependency of the formal specification on the design phase needs some variant of waterfall development method, which isn’t suitable for an agile development. A framework for distributed information systems with mobile architecture, which is introduced in this article, approaches a formal architectural specification in a different way, without the dependency on a design phase. The framework provides an implementation toolkit, which allows an automatic derivation of a formal specification in the process algebra \( \pi \)-calculus in the implementation phase of a project. The independence of formal architectural specification from the design phase can be used for preservation of correct formal description of a system after radical changes in an implementation phase. This can decrease costs of the changes, which are critical, especially in final phases of projects.

Keywords: Formal Approach, Distributed Information System, Mobile Architecture, \( \pi \)-Calculus, Verification

1 Introduction

Contemporary trends in software engineering are influenced by two approaches – a strong component orientation and application of advanced system development life cycle models. The first approach tries to apply advanced system architecture and behavioural patterns, e.g. to achieve a maximum re-usability of existing software products or a hierarchical composition of components according to required functionality. The second approach tries to cope with a variability in user requirements and available resources during a software product development, but without characteristic disadvantages of a waterfall life cycle model. The result of a combination of this two approaches should be high-scalable distributed information systems reusing
Formal Approach to an Architecture of Distributed Information Systems

well-established software products, easily customised and with a very fast and robust life cycle.

Unfortunately, in some cases of information system development, this two approaches cannot be used together. There are critical applications where a component-oriented information system is getting involved, and a formal specification of architecture is necessary. Typical formal approaches to an architectural design require a waterfall-like life cycle model. For this reason, it is very difficult to use the formal approaches in progressive life cycle models (e.g. in an extreme programming methodology) and alter a formal description of architecture with every change in an architectural design.

This paper deals with a framework for distributed information systems with mobile architecture, which can be formally represented as asynchronous network models. The goal of this work is to provide a formal approach to architecture description of information systems with a strong component orientation, without a dependence on the design phase of a project (i.e. the approach suitable for progressive life cycle models). The framework provides an implementation toolkit, which acts as an universal middleware to separate parts of a distributed system by interfaces and allows an automatic derivation of a formal specification in the implementation phase of a project. The result of derivation is a process in a process algebra $\pi$-calculus (known as a calculus of mobile processes), which describes architecture and communication behaviour of the system. A distributed information system implemented using the framework can be formally verified by means of $\pi$-calculus model checker.

The remainder of this article is organised as follows. Section 2 and section 3 introduce a formal base of the framework – a modified asynchronous network model and the $\pi$-calculus principles. In section 4, we describe the framework, its design, support for mobile architecture and an implementation of the framework. Section 5 presents a derivation of formal system architecture description by means of the framework. In section 6 we illustrate through a sample system how the framework can be used for formal specifying an architecture of the system. In section 7 we compare our approach with related work. To conclude we summarise, in section 8, the main contributions of this article and briefly outline the future work.

2 Modified Asynchronous Network Model

Up-to-date information systems are in most cases designed as distributed network systems, i.e. as a network of communicating and partially independent components. Each component performs its specific task, by itself or with help of other components, and all components are connected by a middleware. The communication between components is realised using a message passing mechanism (MPM). Actually, the MPM can be “hidden” on a higher level of abstraction, e.g. behind a
shared memory model or a shared persistent object. Regardless, on a base level, components asynchronously send messages by means of a middleware. In this view, a process in the information system represents a group of components and a scheme of their interaction.

This scheme of communication, on a simplified level, is very similar to a formal asynchronous network model (ANM) with some modifications. The original ANM \cite{Lyn96} consists of a directed graph $G = (V, E)$, where $V$ is set of nodes and $E$ is set of edges. Each node $v_i \in V$ is associated with a process $P_i$ and has in-neighbours and out-neighbours. Each directed edge $e_j \in E$ of the graph is associated with a communication channel, which connects its neighbour nodes in a given direction. Processes and channels are connected using operations \texttt{send} and \texttt{receive} and can be modelled as an arbitrary I/O automaton \cite{Lyn96}.

The \textit{modified asynchronous network model} (MANM, \cite{Ryc06}) introduces two new entities (see Figure 1):

\begin{itemize}
  \item \textbf{a connector} — a communication buffer\(^1\) (formally, an universal reliable FIFO channel \cite{Lyn96}), in the MANM denoted by a special kind of a process (presented as a line), which receives a message from many processes (senders) and delivers it to at most one of many listening processes (receivers).
  \item \textbf{a port} — an interface between a process and a connector in given direction (in the MANM denoted by a black dot and an adjacent oriented edge), which provides operations \texttt{send} and \texttt{receive} and is able to forward sent messages form the parent process to only one connector, but accept messages from many connectors while receiving.
\end{itemize}

The MANM model, in comparison with the original ANM model, provides a better support for modelling of real applications, where the communication between entities is realised through multiple layers. The MANM model divides a system into three layers. The process is in \textit{a process layer} responsible for an application logic, the connector is in \textit{a connector layer} responsible for a reliable communication, and the port acts as an interface between the two layers (in \textit{a port layer}). Furthermore,

\(^1\)the connector is titled as “a link” in \cite{Ryc06}
the described MANM model is translatable into the original ANM where connectors are simple processes forwarding messages according to the previous definition of connector, and ports are communication channels identical to the channels of the original ANM model.

3 Mobile Architecture and $\pi$-Calculus

While the MANM model describes a conceptual framework for the simplified architecture of a distributed information system as a network of communicating processes, a real architecture of contemporary information systems has strong dynamic properties. Current information systems are highly modular, networked, object- and service-oriented, able to dynamically load plug-ins, etc. All these properties are typical for dynamic architecture [MT97]. This architecture supports changes in a system runtime (e.g. creating and destructing processes, or a formation of a new communication scheme). Mobile architecture [MT97] is dynamic architecture with ability to pass entities of the architecture as ordinary messages according to system behaviour.

The mobile architecture of a system can be described effectively by means of a process algebra $\pi$-calculus, known also as “the calculus of mobile processes” [MPW92]. The $\pi$-calculus is an extension of a Calculus of Communicating Systems (CCS), which allows modelling of systems with dynamic communication structures (i.e. mobile processes). It uses only two concepts:\footnote{a parametric process is also titled as “an agent” and the names can be titled according to their meanings (e.g. port/channel, message, etc.)}

\begin{itemize}
  \item a process — an active communicating entity in the system, atomic or expressed in $\pi$-calculus (denoted by uppercase letters in expressions),
  \item a name — anything else, e.g. channel, variable, data, or also a process in a high level view (denoted by lowercase letters in expressions).
\end{itemize}

A process is formally defined in $\pi$-calculus using induction. At first, the process $0$ is a $\pi$-calculus process (null process). If processes $P$ and $Q$ are $\pi$-calculus processes, then following expressions are also $\pi$-calculus processes with given syntax and semantics (the operational semantics of the $\pi$-calculus is described and explained in [MPW92]):

\begin{itemize}
  \item $\pi(y).P$ sends name $y$ via port $x$ and continues as process $P$,
  \item $x(y).P$ receives name $y$ via port $x$ and continues as process $P$,
  \item $\tau.P$ does an internal (silent) action and continues as process $P$,
\end{itemize}
Formal Approach to an Architecture of Distributed Information Systems

- \((x)P\) creates new name \(x\) in a context of process \(P\) and continues as \(P\),
- \(P|Q\) proceeds as parallel composition of processes \(P\) and \(Q\),
- \(P + Q\) proceeds as either process \(P\) or process \(Q\) (a non-deterministic choice),
- \(P(y_1, \ldots, y_n)\) behaves as process \(P\) with substitution \(P\{y_1/x_1, \ldots, y_n/x_n\}\) (the parametric process) where names \(x_1, \ldots, x_n\) occur free in process \(P\).

In the process algebra, an interaction between two processes is formally defined as a reduction of a symbol of an input and output channel by an operation, which symbolises communication. In a system of processes, the reduction means also a transition between two states. For example, the system which is defined in the π-calculus as process \(\pi(y).P|x(z).Q\) (a parallel composition of the process, which sends name \(y\) via port \(x\) and continues as process \(P\), and the process, which receives name \(z\) via port \(x\) and continues as process \(Q\)) can perform a communication step (via port \(x\) from an inside view and as an internal action \(\tau\) from an outside view of the system). After this communication, the system is in a new state, defined as process \(P|Q\{y/z\}\) (all free occurrences of \(z\) in \(Q\) are replaced by \(y\)).

4 The Framework

The framework is based on the formal base, which was described in the previous sections. The framework uses the MANM model to depict a communication structure and a hierarchy of processes, and the process algebra π-calculus for a formal description of dynamic properties of mobile architecture in accordance with system behaviour.

An ideal design and an implementation of the framework has to cope with the next goals:

1. the approach must provide a white-box framework [RJ97] covering principal use-cases and defining interfaces for pluggable objects (particularly at the connector layer), which, in a future, can be enhanced to a black-box framework with a library of concrete components for different applications (particularly for different kinds of connectors),

2. the framework must act as a middleware for interconnecting of information system components and must be able to cope with mobile architecture properties (i.e. especially with sending connections and parts of components as ordinary messages),
3. the framework must be able to reflect a hierarchy of the components, i.e. provide tools for grouping (and hiding) of connected components into a new component with a defined interface,

4. the communication by means of the framework has to be transparent for components in spite of used transmission mechanism (e.g. a direct connection of components via a simple buffer, an inter-process communication by means of an operating system, as well as a network communication according to some kind of service oriented architecture),

5. the designer of a system using the framework has to be implicitly forced to design the system as the MANM model (i.e. using process-port-connector pattern) and the framework has to restrain changes in an implementation phase from going beyond the scope of the MANM and the mobile architecture.

4.1 Design of the Framework

The design of the framework is based on the MANM model and the process algebra $\pi$-calculus. The framework uses the MANM model for decomposing a component interaction into three layers (in a vertical view) and for defining a hierarchy of components (in a horizontal view), and the $\pi$-calculus for describing system behaviour related to communication.

In the vertical view, there are a process layer, port layer and a connector layer. The process layer contains atomic processes, which are implemented directly by the information system without an explicit support of the framework, and composite processes, which represent a group of processes connected together by the framework. The port layer is an interface which is provided by the framework to processes. The connector layer is a low-level communication support, a middleware, fully handled by the framework.

In the horizontal view, the atomic and composite processes from the process layer are used for defining a hierarchy of components. The atomic process uses the framework only for communication with another processes – i.e. from the framework view, the atomic process is "a blackbox". For the purpose of communication, the framework provides an atomic process interface for transmission of messages of specified type. According to MANM model, the atomic process is an indivisible process, ports are its interfaces and the framework provides a connector for the communication. On the contrary, the composite process represents a group of processes connected together by the framework (by connectors) via provided interfaces (via ports). The

\footnote{without a knowledge of used formalisms}

\footnote{the designer needn’t be forced to use the mobile architecture, because “ordinary” (static and dynamic) architecture is a subset of the mobile architecture}
The designed framework must support a composite process management and behaviour. It has to provide a mechanism for attaching and detaching of processes into a composite process, interfaces (ports) for processes outside the composite process and an execution support for inside processes.

The basic structure of the framework is described in the class diagram in Figure 2. The framework is designed as a complete middleware with interface ProcessImpl, through which a component of the information system can be attached to the framework. The interface ProcessImpl is “a hook”, which the component should implement. It provides public methods run for execution of the component and build for initialisation of the component. An atomic process or a composite process representing the component calls these methods, when execution or initialisation of the component is needed.

At the process layer, the interface Process represents a common interface for an atomic process (class AtomicProc) and composite process (class CompositeProc). It sets protected methods run to “execute” the atomic or composite process and build
to prepare the process for execution. Methods run and build are called directly by
the framework, when execution or creation of the process is needed. These methods
do required internal actions in the framework and then call relevant methods of
associate implementation of interface ProcessImpl, i.e. methods described in the
previous paragraph. The internal action for run of CompositeProc can be, for
example, calling of method run for each internal process of the composite process.

The AtomicProcPort, CompositeProcRelAY and RelayPort are at the port layer. The RelayPort acts as a proxy between processes inside and outside a composite
process in both directions. The Connector and auxiliary class ConnectedPort are at
the connector layer, where a port and connector communicate according to a design
pattern observer. Real implementation of the class Connector should be adapted
for a real communication mechanism, such as a connection via a shared memory or
a network connection.

The Figure 3 shows a sequence diagram of communication between two compo-
nents of information system (i.e. two atomic processes of the framework). The se-
quence is initialised by process receiver, which calls method receive of port portR
and waits for finishing. After process sender calls method send of its AtomicProc-
Port port portS, a message (a parameter the method) is prepared and a connector
connSync is notified by calling of method updateFromPort (according to a design
pattern “observer”). The connSync notifies all attached observers (again according
to a design pattern “observer”). The usage of the design pattern “observer” on both
sides of Connector in the communication allows, that Connector is able to pass the
message without an internal message buffer\(^5\). Some of the notified observers, ports
which are receiving, call method passSent of the connector. Only for one of them,

\(^5\)without loss of generality, because the implementation with a message buffer is also possible
port portR of process receiver in the sequence diagram, the connector calls method passSent of the port portS. The return value of this method contains the message and is passed via the connector to portR (again as return value). Then method send has finished (the message has to be sent) and method receive has finished (the message has to be received). The transfer is complete.

4.2 Support for Mobile Architecture

As it was mentioned in the Section 3, the mobile architecture extends the dynamic architecture with ability to pass entities of the architecture (i.e. processes and connections) as ordinary messages. This feature can be expressed in the process algebra π-calculus, which is used as a formal base for behaviour of information systems supported by the framework.

The passing of communication channels is available directly in the π-calculus (see “a scope extrusion example” in [MPW92]). In the following transition (1) process Q sends port b via channel a and process P receives sent port b via channel a as x. Then, process Q is finished (it is null process 0), process P continues as process P' and process R is unchanged. The result of this transition is passing of the port b from process (b|Q|R), where b is hidden and therefore not accessible from an outside of the process (a scope is “extruded” [MPW92]).

\[
\begin{align*}
(a)(a(x),P') & \rightarrow (b)(\pi(b).0 \parallel b(z).R') \Rightarrow (b)(P'{b/x} \parallel b(z).R') \\
\end{align*}
\]

The passing of π-calculus processes isn’t directly possible in the basic π-calculus formalism, but it is available as an indirect representation by means of the passing of communication channel, where the channel is linked to the “passed process” (see “an executor example” in [MPW92]). The indirect representation is demonstrated in transition (1) with process P’ = π.0, where communication via z execute process R, which is located in a different part of the system (i.e. R is executed “instead of” process P). A critical part of the passing of π-calculus processes is preservation of a process environment (i.e. communication channels, which are connected to the process). The framework has to cope with this “environment corruption”, as it will be shown in the rest of this section.

The Figure 4 describes an implementation of the transition (1) in the framework. In this example, process Q, which is inside a composite process together with process R (i.e. the composite process represents subprocess (b|Q|R)), is connected via port b to process R. The processes Q and R are interconnected via their ports b and the connector W. In addition, the process Q is connected to a composite process boundary (a port “relay”, which forwards messages between inside and outside processes) via connector V. The external process P is connected to an outside of the composite process boundary to the same port as process Q.
Now, suppose that process $Q$ sends its port $b$ towards process $P$ via port $a$. In fact, it sends "a connection from port $b$" (see step (1) in the Figure 4). The connection is passed via connector $V$ to the composite process boundary (step (2)) and in the next step, it comes via connector $U$ to process $P$ (steps (3) and (4)). Because the received connection has corrupted a relation to the original environment, the process $P$ makes a query to a connector manager (step (5)) to resolve a scope of the received connection. The connector manager recreates the environment of the connection, which results in a new connection from process $P$ towards connector $W$ (step (6)).

The connector manager provides a management of the connector layer including creating, destructing, connecting and disconnecting of connectors, recreating of an environment of ports after passing of the ports, etc. The connector manager is purposely designed separately and its final design and implementation is the aim of the future work.

4.3 Implementation of the Framework

Generally, implementation of the framework isn’t limited by a particular programming language. Regardless, a good choice of the language can make an implementation easier. Desirable features, which the selected programming language should fulfil, are following:

- support for *object-oriented programming* (this is an easy condition),

- support for *concurrent programming* (multi-threaded applications, synchronisation, etc.) – the goal of the framework is to enforce implicit design of a distributed information system as system of asynchronous concurrent processes (the MANM model), i.e. the support of a concurrence directly in the framework is very useful,
• platform independence and a support for serialising objects – this can be useful for an implementation of the mobile architecture features, i.e. for passing of processes and connections, as it was mentioned in the Section 4.2.

The Java programming language fulfils all described features, therefore it was selected for the implementation of the framework, which is part of ongoing work.

5 Derivation of a Formal Description

Automatic derivation of a formal description of an architecture and behaviour is the main aim of the framework. The framework specifies an environment for an implementation of a distributed information systems, which forces designers (implicitly) and coders (explicitly) to respect the formal base of the framework and where the framework itself acts as a middleware of the implemented system. Therefore, initial configuration and changes of architecture of the system are realised only by means of the framework, which is able to describe them in the process algebra π-calculus.

Actually, the framework isn’t able to derive a formal description of an information system as a whole only from its implementation. It is because of atomic process behaviour, which is hidden in implementation of a system component. The atomic process is “a blackbox” from the framework point of view. For that reason, a description of the atomic process behaviour has to be done manually. Thereafter, the framework can automatically generate a composite process formal specification, which is based on a formal specification of internal processes and communication behaviour of the composite process provided by the framework. After recursive derivation of formal descriptions of composite processes through the hierarchy in the information system architecture, the framework is able to derive correct formal description of the system as a whole (as the system is a composite process).

Now, suppose we have a system implemented using the framework. At first, for each atomic process there is given its description of behaviour in π-calculus (behaviour of a AtomicProcess instance). Then, a port \( p \) (an instance of AtomicProcPort) of a process in the framework is expressed as two channels in π-calculus: \( p_{in} \) for receiving and \( p_{out} \) for sending. A connector (an instance of Connector) in the framework, which transmits messages from ports represented in π-calculus as \( q_{1_{out}}, \ldots, q_{m_{out}} \) to ports represented as \( p_{1_{in}}, \ldots, p_{n_{in}} \), can be expressed as a repeated non-deterministic choice of two opposite channels and communication between them:

\[
\text{Connector}(p_{1_{in}}, \ldots, p_{n_{in}}, q_{1_{out}}, \ldots, q_{m_{out}}) = \sum_{i=1}^{n} \sum_{j=1}^{m} q_{j_{out}}(x). p_{i_{in}}(x).
\]

The non-determinism in the definition (2) is necessary, because the connector
isn’t able to determine, which of connected channels (ports) are ready for receiving and there can be a failure: a message can be sent to an incorrect channel of an inactive process while another process is ready to receive it.

The last two entities of the framework, which have to be translated into the $\pi$-calculus, are the composite process (instance of $\text{CompositeProcess}$) and the port of a composite process (instance of $\text{CompositeProcRelay}$). In the $\pi$-calculus, the composite process can be defined as a parallel composition of its internal processes. The port of a composite process acts as a proxy transferring messages from a connector outside towards a connector inside the composite process. This action can be translated into the $\pi$-calculus simply as a direct connection of processes, which represent the outside and inside connector, via their channels. In practise, it means substitution (renaming) of the channels to the same name or defining the composite process as a parametric $\pi$-calculus process with ports of inside connectors as parameters.

### 6 Sample System

As an example of a distributed information system, we will demonstrate design and description of a log server. The system consists of a client, which produces some events and sends them to a server, the server, which provides an interface for logging events incoming as messages and for providing log dumps, and second client, which makes and sends a requests for a log dump to the server a receives results. A system architecture (see Figure 5) consists of process $A$ (the first client), process $B$ (a part of the server, which creates process $P$ on demand of process $A$), process $C$ (a part of the server, which stores logs and provides process $D$ with stored logs on demand), process $D$ (the second client) and process $P$, which receives events from process $A$ and selects only crucial events (for our example, all events are selected).
The system behaves as follows (the described steps correspond to numbers in the Figure 5):

1. the process $A$ sends a message containing\(^6\) port $b$ via port $c$ to port $m$ of server, which forwards the message towards port $e$ of process $B$,

2. the process $B$ creates a new process $P$ with ports $f$ and $g$, where port $g$ is connected to port $h$ of process $C$,

3. then process $B$ sends port $f$ via port $d$, which has received in the message, to port $b$ of process $A$ and when process $A$ receives the message, it connects\(^7\) its port $a$ to the port $f$,

4. when process $A$ makes an action, it sends an announcement (a message) via port $a$ to port $f$ to process $P$,

5. the process $P$ can decide if the action from the received announcement via port $f$ is important or not, and forwards it via ports $g$ to port $h$ to process $C$, which stores the received message,

6. the process $D$ can independently of preceding steps send a message containing port $l$ (a request for stored logs) via port $k$ to port $n$, which forwards it towards port $i$ of process $C$,

7. after receiving the message via port $i$, process $C$ sends stored messages (events from the log) via received port $j$ to port $l$ of process $D$.

The described system can be implemented by means of the framework. Only formal specifications of atomic processes (components) have to be defined in the design phase, but a formal description of the server (i.e. composed process $S$) and the system as a whole can be derived automatically. Any later changes in the system architecture, out of the atomic processes, can be projected into the formal description by the framework. The atomic processes can be expressed in the $\pi$-

\(^6\)when processes establish a bidirectional connection dynamically (it isn’t a part of an initial architecture description) a port for a reverse direction must be sent

\(^7\)all used connectors are parts of an initial configuration of the system or they are created dynamically, e.g. a connector between $d$ and $b$, as was described in Section 4.2
14  Formal Approach to an Architecture of Distributed Information Systems

calculus as follows:

\[ A(c) = (b)(\bar{c}(b).b(a).A'(a)) \]  

\[ A'(a) = \text{Action}(x).\bar{a}(x).A'(a) \]  

\[ B(e, g) = e(d).(B(e, g)|B'(d, g)) \]  

\[ B'(d, g) = (f)(\bar{d}(f).P(f, g)) \]  

\[ C(h, i) = C_1(h)|C_2(i) \]  

\[ C_1(h) = h(m).\text{Store}(m).C_1(h) \]  

\[ C_2(i) = i(j).\text{LoadAll}(m).\bar{j}(m).C_2(i) \]  

\[ D(k) = (l)(\bar{k}(l).l(m).\text{Analyse}(m).D(k)) \]  

\[ P(f, g) = f(x).\bar{f}(x).P(f, g) \]

Processes \text{Action}(x), \text{LoadAll}(m), \text{Store}(m) and \text{Analyse}(m) represent parts of internal behaviour of the parent processes. The first two (in (4) and (9)) can be defined as a creation of a new “name” (i.e. \text{Action}(x) = (x) and \text{LoadAll}(m) = (m)) and the next (in (8) and (10)) can be defined as the null process (i.e. \text{Store}(m) = \text{Analyse}(m) = 0), see Section 3.

On the basis of the definitions of the atomic processes and implementation of the system according to the Figure 5), the framework is able to derive a formal description of the server \( S \) and the System as a whole (for a definition of connector \( \text{Con}(p_{1in}, \ldots, p_{nin}, q_{1out}, \ldots, q_{nout}) \) see Section 5):

\[ S(m, n) = (e)(g)(h)(i)(\text{Con}(m, e)|B(e, g)|\text{Con}(g, h)|C(h, i)|\text{Con}(n, i)) \]  

\[ \text{System} = (m)(n)(c)(k)(S(m, n)|\text{Con}(c, m)|A(c)|\text{Con}(k, n)|D(k)) \]

It is obvious, that the derived formal specification is very easy and the descriptions of the atomic processes are much harder. Regardless, the atomic processes are designed as independent components and can be combined into more complex systems.

An example of a verification of a similar system can be found in [Ryc06].

7 Related Work and Discussion

In the literature, there have been proposed several approaches, which focus on a formal description of an information system architecture by means of some formalisms – there is a model NOAM (Net-based and Object-based Architectural Model, [DLE97]), which uses a hierarchical object approach with a notation and semantics of Petri-nets for the modelling of real-time systems, a temporal logic approach to the specification of reconfigurable component-based systems [AM02], which
is able to capture a hierarchical structure and run-time changes of a system with the dynamic architecture by means of a linear temporal logic, or the Wright architectural specification language [AG96], which uses an algebra of communicating sequential processes (CSP) for description of behaviour of processes in dynamic architecture and introduces a new conception of architectural styles (i.e. patterns of components with specified integrity constraints and interfaces, see [AG96]).

Overall, all these approaches define some kind of an architecture description languages (ADLs, [MT97]), which allow a formal specification of the system architecture using a textual or graphical description in the design phase of a project. Some of them haven’t been explicitly named as “ADLs” despite their characteristics, which fulfil a definition of ADL [MT97].

The ADL languages have been designed in order to cope with architectures of specific application domains (e.g. to support run-time systems, systems using common object request broker architecture “CORBA”, etc.). This application oriented design, together with specialised modifications of used formalisms, leads to mutually incompatible languages, with a complicated syntax and semantics. Besides, in many cases, application of an ADL brings an unreasonable complexity for systems with a simple architecture. The second generation of ADLs, including ACME [GMW00] and Unified Modelling Language version 2 (UML2)\(^8\) has brought an unification of an ADL implementation and ADLs are ready for an appliance in real information systems. There are also available several converters from the second generation ADLs (especially form the ACME) to a first generation ADL with a formal base, which allows description of an architecture in a selected formalism.

We believe that one of reasons why current ADLs aren’t used for formal architecture description in practice is their strong dependence on description in the design phase of a project. The formal description in an ADL is derived from an architecture description in the design phase. For that reason, the ADLs required some variant of a waterfall development method and, in many cases, they aren’t be suitable for agile development methods. Our framework approaches formal architectural specification in a different way.

Unlike ADLs that provide tools for formal specification during design phase, the framework provides an implementation toolkit, which allows automatic derivation of formal specification in the implementation phase of a project. The independence from the design phase of a project can be used for preservation of correct formal description of a system after radical changes in the implementation phase. This can decrease costs of the changes, which are critical, especially in final phases of projects. Such approach probably hasn’t been described in the literature yet.

Despite the independence from the design phase of a project, the framework

---

\(^8\)despite the fact that the UML2 implements only a subset [AGM05] of ADL properties and without an explicit formal base
requires formal description of atomic processes in the process algebra $\pi$-calculus (see Section 5). For that reason, we expect usage of the framework together with another formal approach for initial formal description of system components in the design phase. Thereafter, the framework itself can be used for verification of the formal description in late phases of a project. As the formal approach at "the design level", there can be used $\pi$-ADL [Oqu04], a part of the ArchWare European Project, which is based on the process algebra $\pi$-calculus.

8 Conclusion and Future Work

This article, there is described the present state of design and implementation of the framework for distributed information systems with a strong formal base. The goal of the framework isn’t an outline of some formal approach or a new tool for architectural design, but the design of a mechanism for automatic derivation of formal architectural specification from an implemented system.

The framework acts as a middleware. It it splits a distributed information system into components (processes), provides an interface of components (ports) and implementation of a communication layer (connectors). The resulting formal specification of the system architecture can be used in model checking (verification of correctness), simulation of many concurrent runs of a system components, etc.

An ongoing work is related to the final design and implementation of the framework including the connector manager (see Section 4.2). There are technically difficult parts, which include correct implementation of component behaviour with a concurrency and passing of ports and processes with preservation of their state. The future work will be related to application of the framework to practical case studies, a development of supporting tools and connection to another formal approaches.

This work has been supported by the Grant Agency of Czech Republic grants No. 102/05/0729 “A Framework for Formal Specifications and Prototyping of Information System’s Network Applications”.

References


