Distributed Information System as a System of Asynchronous Concurrent Processes

Marek Rychlý and Jaroslav Zendulka

Faculty of Information Technology
Brno University of Technology
Department of Information Systems
{rychly, zendulka}@fit.vutbr.cz

Abstract. Nowadays enterprise information systems are designed as distributed network systems, where existing information systems and new components are connected together via a middleware. In most cases, architectures of the systems can be described informally or semiformally by means of common design tools. But there are also critical applications where an information system is getting involved, and a formal architecture specification is necessary. This paper describes a design of a framework for distributed information systems with a mobile architecture and an outline of its implementation. The framework provides an automatic derivation of a formal specification from an implementation of system, without an explicit formal description in a design phase of project. The derived specification can be used for a quick formal proof of correctness after radical changes in an implementation phase, without a maintenance of a formal design.

1 Introduction

Nowadays information systems are in most cases designed as distributed network systems. A globalisation in a sphere of business creates need for decentralised systems with a correct data distribution, distributed processing, reservation of resources and a reliable communication infrastructure. The distributed information systems are designed as a network of communicating and partially independent components where each component performs its specific task, by itself or with help of other components. In this view, a process in the information system represents a group of components and a scheme of their interaction.

An architecture design of a distributed information system and its implementation can be very complex and difficult. Formal approaches can eliminate most failures by model checking, but they require a formal specification of architecture in a design phase of project. Unfortunately, many enterprise information systems aren't build "from scratch", they are developed as confederations of existing information systems reusing a well-established software products. In this case, a designer of a distributed information system hasn't a full control over an architecture design and a formal model often doesn't correspond to the real implementation of information system.

In this paper, there is described a framework for distributed information systems with a mobile architecture, which can be formally represented as asynchronous network models. The goal of this work is the abstract object-oriented architectural model, which is compatible with a calculus of mobile processes, known as the π-calculus. The framework is an implementation of the architectural model with the formal base and it represents an universal middleware to separate parts of a distributed system by interfaces with the preservation of formal properties. A distributed information system implemented using the framework can be formally verified by means of π-calculus model checker.

2 Formal Base

A typical distributed information system is designed as a group of partially independent components, where each component performs its specific task and all components are connected by a middleware. The communication between components is realised using a message passing mechanism (MPM). The MPM can be "hidden" on a higher level of abstraction, e.g. behind a shared memory model or a shared persistent object, but in fact, on a base level, components asynchronously send messages by means of a middleware (i.e. it is the MPM).

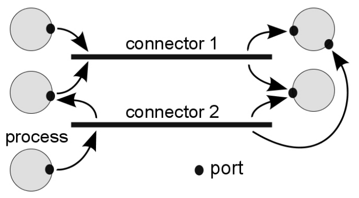

Fig. 1. Modified asynchronous network model (MANM).

This scheme of communication, on a simplified level, is very similar to a formal **asynchronous network model** (ANM) with some modifications. The original ANM [1] consists of a directed graph of *processes* (nodes) connected by *communication channels* (edges of the graph). Both, processes and channels, can be described as an arbitrary I/O automaton, connected using operations *send* and *receive*. The modified ANM (MANM, see [2]) introduces two new entities (see Figure 1):

a port — an interface between a process and a connector in given direction, in the MANM denoted by an oriented edge,

a connector — a communication buffer[1], in the MANM denoted by a special kind of process (presented as a line), which receives messages from ports leading to the connector and forwards messages towards ports leading from the connector (but one message can be forwarded to one port only).

[1] the connector is titled as "a link" in [2]

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 original ANM.

While the MANM model describes a conceptual framework for the simplified architecture of a distributed information system as a network of communicating processes (for detailed description see [2]), a real architecture of contemporary information systems has strong dynamic properties. The *dynamic architecture* allows creating and destructing processes and a formation of new communication scheme in a system runtime. On a higher level, there is a *mobile architecture*, which is a dynamic architecture with ability to pass entities of architecture (i.e. processes and connections) as ordinary messages. The mobile architecture of system can be described formally by means of **process algebra π-calculus**, known also as "the calculus of mobile processes" (see [3]). The π-calculus uses only two concepts[2]:

a process — an active communicating entity in the system, atomic or expressed in π-calculus (denoted by uppercase letters in expressions),
a name — anything else, e.g. channel, variable, data, or also a process in a high level view (denoted by lowercase letters in expressions).

A process is formally defined in π-calculus using induction. At first, the process 0 is a π-calculus process (null process). If processes P and Q are π-calculus processes following expressions are also π-calculus processes with given syntax and semantics (the operational semantics of the π-calculus is described and explained in [3]):

- $\overline{x}\langle y\rangle.P$ sends name y via port x and continues as process P,
- $x(y).P$ receives name y via port x and continues as process P,
- $\tau.P$ does an internal (silent) action and continues as process P,
- $(x)P$ creates new name x in a context of process P and continues as P,
- $[x = y]P$ proceeds as P if names x and y are identical, else behaves like a null process,
- $P|Q$ 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 symbol of input and output channel by an operation, which symbolises a 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 $\overline{x}\langle y\rangle.P|x(z).Q$ (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

[2] a parametric process is also title as "a agent" and the names can be titled according to their meanings (e.g. port/channel, message, etc.)

communication step (via port x from an inside view and as an internal action τ 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).

3 Design and Implementation of the Framework

The framework for implementation of distributed information systems as systems of asynchronous concurrent processes is grounded in the formal base, which is described in the previous chapter. The framework uses the MANM model to catch a communication structure (a vertical view) and a hierarchy of processes (a horizontal view), and the process algebra π-calculus for a formal description of system behaviour. After a short description of the vertical and horizontal views, this part of the paper will aim at analysing several important properties, which are specific to mobile architectures and which the framework has to deal with.

3.1 Vertical View = Modified Asynchronous Network Model

The framework uses the modified ANM model for decomposing a component interaction into three layers in **a vertical view**: a process layer, port layer and a connector layer. In this view, behaviour of the *process layer* is implemented locally in components of the system, without an explicit communication support of the framework. The *port layer* is an interface, which is provided by the framework to a component of the system. The *connector layer* is a low-level communication support, a middleware, fully handled by the framework (a more detailed description of the three-layer vertical view is in [2]).

3.2 Horizontal View = Atomic and Composite Process

In **a horizontal view**, there are designed two types of processes: an atomic process and a composite process. The *atomic process* is a component of a distributed information system, which isn't implemented by means of the framework – i.e. from the framework view, the atomic process is "a blackbox", which uses the framework only for a communication with another processes. For this purpose, the framework provides the atomic process an interface for a 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. Because the atomic process is "a blackbox", the framework isn't able to derive a formal description of communication behaviour of the atomic process and the description has to be done manually.

The *composite process* represents a group of processes, connected together by the framework (by connectors) by provided interfaces (ports). A purpose of the composite process design is to allow a hierarchical composition of a distributed information system. The framework fully implements a composite process management and behaviour – it provides a mechanism for attaching and detaching

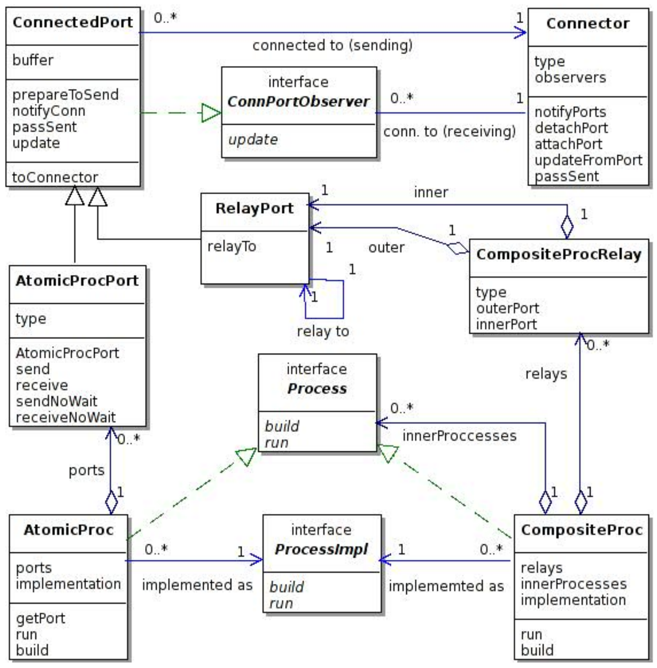

Fig. 2. An outline of class diagram with the process, port and connector.

of processes into a composite process, an interface (ports) for processes outside of the composite process and execution support for inside processes. The framework supports also an automatic generation of a composite process formal specification in the π-calculus, which is based on a formal specification of internal processes and communication behaviour of the composite process provided by the framework.

The Figure 2 shows a part of a class diagram of the framework design. The `Process`, `AtomicProc`, `CompositeProc` and `ProcessImpl` are in the process layer. In `ProcessImpl` interface, there are provided "hooks" for an initialisation of a process (i.e. building ports and an establishment of an initial connection between the process and its environment; the method `build`) and for implementation of the process (a main implementation of an atomic process and an additional[3] implementation of a composite process; the method `run`). The `AtomicProcPort`, `CompositeProcRelay` and `RelayPort` are in the port layer. The `RelayPort` acts as a relay/proxy between processes inside and outside of a composite process in both directions. The `Connector` and the auxiliary class `ConnectedPort` are in the connector layer, where a port and connector communicate according to a design patter observer.

[3] besides of a communication support, which is implicitly provided by the framework

3.3 Behaviour = Typed π-Calculus

As it was mentioned in the previous paragraphs, **behaviour** of a system implemented using the framework can be formally described by means of the process algebra π-calculus (in fact, it is a typed π-calculus, see [4]). A basic process of translation from an implementation of distributed information system in the framework to a process of π-calculus was described in [2]. In this paper we will deal with an application of specific properties of the mobile architecture, i.e. forwarding of communication channels and processes as ordinary messages.

The *passing of communication channels* is available directly in the basic π-calculus as a transition (see "a scope extrusion example" in [3]):

$$(a)(\overbrace{a(x).P'}^{P} \,|\, (b)(\overbrace{\overline{a}\langle b\rangle.0}^{Q} \,|\, \overbrace{b(z).R'}^{R})) \xrightarrow{a} (b)(P'\{b/x\} \,|\, \overbrace{b(z).R'}^{R}) \tag{1}$$

In this transition a π-calculus port b is passed from subprocess $(b)(Q|R)$ (where b is hidden and therefore not accessible from an outside of this subprocess) to the process P' via channel a between Q and P.

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 a passing of communication channel, where the channel is linked to the "passed process" (see "an executor example" in [3]). The indirect representation is demonstrated in transition (1) with process $P' = \overline{z}.0$, where a communication via z executes 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 a 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 the chapter.

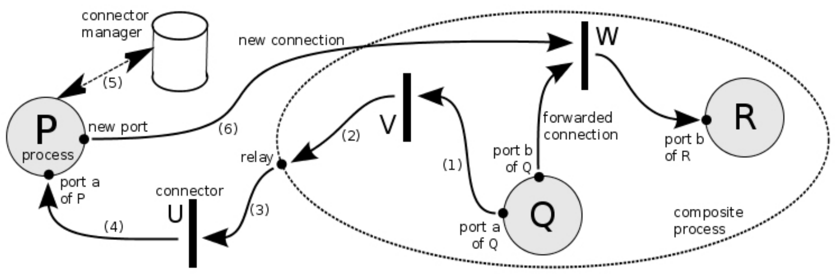

Fig. 3. Passing of port b from composite process $Q|R$ to process P in the framework.

The Figure 3 describes implementation of the transition (1) in the framework. In this example, process Q, which is inside of 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 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) on the Figure 3). This 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 component manager (step (5)) to resolve a scope of the received connection. The component manager recreates the environment of the connection, which results in a new connection from process P towards connector W (step (6)). The framework provides a local synchronised instance of connector W, although it is enclosed into the composite process.

4 Related Work

There are many different approaches to a formal architecture design, which are based on different formalisms (e.g. on Petri-nets, temporal logics or process algebras). Overall, all these approaches define some kind of an architecture description languages (ADLs, see [5]), which allows a formal specification of the system architecture using a textual or graphical description in a design phase of a project. For that reason, the ADLs may require some variant of a waterfall development method and in specific cases, they can't be suitable for agile development methods.

Most of the up-to-date ADLs are designed for the dynamic architecture, but in many cases, a direct application of an ADL can bring an unreasonable complexity for a simple system architecture. In the mobile architecture, the complexity of a distributed system is higher by several orders and a formal model can be the best way to catch the system complexity. In the ArchWare European Project, there was realised an ADL for mobile architectures, a π-ADL (see [6]), which is based on the process algebra π-calculus. A design of the π-ADL is similar[4] to our framework due to the same formalism, in spite of the fully independent development of these approaches.

Our framework approaches a formal architecture specification in a different way. Unlike ADLs that provide tools for a formal specification during design phase, the framework provides an implementation toolkit, which allows an automatic derivation[5] of a formal specification in an implementation phase of a project. Such approach probably hasn't been described in the literature yet.

A merit of our approach is the independence from a design phase of a project, which can be used for a preservation of a correct formal description of a system

[4] the similarity can be utilised, e.g. for sharing of verification tools

[5] from a manual formal description of atomic processes and implementation by means of the framework

after radical changes in an implementation phase. This can decrease costs of the changes, which are critical, especially in final phases of projects.

5 Conclusion and Future Work

This paper describes a design of the framework for distributed information systems with a strong formal base. A goal of the framework isn't an outline of some formal approach or a new tool for an architecture design, but the design and implementation of mechanism for an automatic derivation of a formal architecture specification from an implemented system.

The framework acts as a middleware, i.e. it splits a distributed information system into components (processes), provides an interface of components (ports) and an implementation of a communication layer (connectors). The resulting formal specification of the system architecture can be used in a model checking (a verification of correctness), a simulation of many concurrent runs of a system components, etc. (an example can be found in [2]).

An ongoing work is related to completing design and to implementation of the framework. There are technically difficult parts, which include correct implementation of component behaviour with a concurrency, passing of ports and processes with a preservation of their state and context and a transparent presence of connectors in extensive networks (i.e. on many locations). Future work is mainly related with an 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. 102/05/0723 "A Framework for Formal Specifications and Prototyping of Information System's Network Applications".

References

1. Lynch, N.A.: Distributed Algorithms. Morgan Kaufmann (1996)
2. Rychlý, M.: Towards verification of systems of asynchronous concurrent processes. In: Proceedings of 9th International Conference ISIM'06. (2006) 123–130
3. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes, part I/II. Journal of Information and Computation **100** (1992) 41–77
4. Pierce, B.C., Sangiorgi, D.: Typing and subtyping for mobile processes. Mathematical Structures in Computer Science **6**(5) (1996) 409–454 An extract appeared in *Proceedings of LICS '93*: 376–385.
5. Medvidovic, N., Taylor, R.N.: A framework for classifying and comparing architecture description languages. In: Proceedings of the 6th European conference held jointly with the 5th ACM SIGSOFT international symposium on Foundations of software engineering, Springer-Verlag New York, Inc. (1997) 60–76
6. Oquendo, F.: π-ADL: an architecture description language based on the higher-order typed π-calculus for specifying dynamic and mobile software architectures. ACM SIGSOFT Software Engineering Notes **29** (2004) 1–14