Programming smart environments using π-calculus

Geetha lekshmy Va,*, Jasmine Bhaskar

Abstract

In the realm of Internet of Things (IoT), the complexity of designing ubiquitous and smart systems with dynamically evolving structure has grown to an extent where system modelling and verification has become a real hurdle. Though, a lot of innovations have taken place in this field, there are no good mechanisms for describing and modelling smart environments in a formal manner. π-calculus, proposed by Milner is a formal language which provides strong mathematical base that can be used for modelling and verifying system requirements. In this paper, we intend to offer a contribution towards understanding the usefulness of π-calculus as a language for programming a special kind of ubiquitous application: smart environments. Here we propose a model that includes a type checker for π-calculus and an application that executes π-calculus statements and generates the graphical representation of the smart environment represented by the π-calculus statement.

Keywords: smart environments; π-calculus; programming language

1. Introduction

Internet of Things (IoT) poses an emerging trend of sensing, computing and communication. As a result modern homes and offices may be rapidly transformed into smart environments enriched with more and more intelligent devices. Though IoT enable the implementation of a new generation of ubiquitous applications, as a variety of sensors, appliances and communication devices are involved, it raises the development complexity.

“A smart environment is a small world where sensor enabled and networked devices work continuously and collaboratively to make lives of inhabitants more comfortable”3. The smart environment application mainly focuses on supporting the users by sensing the environment and generating appropriate actions to the changing environment. A smart environment is populated with sensors, actuators and processing elements that may get added or removed on the fly, giving it a dynamic structure. Sensor is any physical equipment influenced by the environment or human behavior like temperature sensor, light sensor etc. An actuator is any device which alters the

*Corresponding author. Tel.: +91 9747313061
ggeethalekshmy@am.amrita.edu

1877-0509 © 2015 Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

Peer-review under responsibility of organizing committee of the International Conference on Information and Communication Technologies (ICICT 2014)
doi:10.1016/j.procs.2015.02.158
environment, air conditioner, lamp, alarm etc. Sensors monitor the environment, collect data and send to the processing elements. The Processing elements work on the data; make decisions and sends commands to the actuators. As each of these components is independent, with their own working capabilities, it is difficult to provide seamless and distraction free support for the smart environment users. The behavior of smart environments becomes complex due to the intricate communication between these components and as a result, the probability of design errors may increase. Also, testing smart environment can be complex, time consuming and thus costly if done in real environment with real users. So there is a real need for a modelling technique that tests the smart environment structure before deployment. Hence providing formalisms or techniques that will enable the efficient design of correct and well-functioning systems despite their complexity appears to be a major challenge. Also, the tremendous increase in smart environment projects compel people to think of efficient, simple and novel methods for smart environment representation.

Formal methods seems to be a promising way of modelling complex system as they can model the requirements in the form of properties by using logic based on mathematics and formally verifying the modeled system against the requirements. They can trace back the errors and can help in fixing them at early design stages. Formal methods can be used for modelling smart environment since faults can be reduced and correctness of the system can be mathematically proved. π-calculus is a mathematical model for communicating systems. The basic computation step in π-calculus is transfer of communication link between two processes. The recipient can use the link for further interaction with other processes. This feature makes π-calculus suitable for modelling dynamic systems. This paper, intends to offer a contribution towards understanding π-calculus as a language for programming smart environments. Here, we propose an approach for the development of a software application for the execution and observation of smart environment based on π-calculus. This study intends to prove that well typed π-calculus statements can be used to program correct smart environments. The proposed model comprises the ideas for type-checking π-calculus statements, executing the π-calculus statement as well as graphically representing the same. In this paper we use a π-calculus based language where the symbols and typographical expressions in π-calculus process, are denoted by ASCII words.

Lack of proper type checker and parser for π-calculus and lack of tool support for π-calculus simulation are major motivators towards this study. Novelty of this paper is the idea of using π-calculus as a language for modelling smart environment and validating its structure. The smart environment functionality is encoded in π-calculus and its corresponding code is generated and executed for checking the correctness of process interactions and to obtain a graphical view of the new system.

2. Related works

Even though various approaches have been proposed for modelling certain aspects of smart environments, there has not been much work in describing formal models of smart environments. Foundational research on the ambient calculus by Cardelli and Gordon shows process calculi as a promising approach to modelling mobile computation. Ranganathan and Campbell have proposed a model to make provably correct pervasive computing environment using ambient calculus. Also some work related to modelling smart environments using petri nets is done at Software Engineering Competence Center, ITIDA, Egypt. In this petri-net controllers are used to formally capture the specifications of smart environment.

Though, π-calculus has aroused intense interest in research to study its applicability and extensibility, at present there is no work related to its applicability to model/program smart environments. But it has been shown that π-calculus is powerful enough to model various mobile communicating systems like internet and Business Process Management systems.

Puhlmann discusses the applicability of π-calculus as a formal foundation for Business Process Management (BPM). π-calculus is the basis of several experimental programming languages such as Pict, Occam pi and also the basis for Business Process Execution Language (BPEL). The paper shows how the correspondence between π-calculus and BPEL activities helps in generating BPEL specifications from π-calculus. Business Process Execution Language (BPEL) is used to express web services orchestrations. A BPEL process defines how multiple service interactions with partners are coordinated internally to achieve a business goal (orchestration). A WS orchestration shows Web Services running in parallel and this can be represented by parallel construct in π-calculus.

The study related to modelling Web Service Interaction, explains the use of π-calculus in modelling web service interaction by modelling the dynamic linking behavior of processes within service oriented architectures. In this work, process in π-calculus is used to express web services and process channel is used to express message during
service interactions. \( \pi \)-calculus can also be used for formal modelling and verification of web service composition. The modelling and verification of the composition of web services are very difficult as this involves communication and collaboration of several Web services. The validity of the process and the property of non-deadlocks must be simulated and proved before being implemented formally. A Web service composition modelling method based on \( \pi \)-calculus is presented where a Web service composition is modelled as a composition of a set of concurrent web services, using the concurrent operators provided by \( \pi \)-calculus. On the basis of this specification, the correctness of web service composition process is verified.

3. Background

3.1 \( \pi \)-calculus

\( \pi \)-calculus, an extension of CCS, was developed by Milner, Parrow and Walker. “It is the theory of mobile systems that provides a conceptual framework for understanding mobility, and mathematical tools for expressing systems and reasoning about their behavior”\(^{12}\). In \( \pi \)-calculus, we can represent processes, parallel composition of processes and non-determinism. \( \pi \)-calculus processes which execute in parallel can be given by \( xy.P|x(u).Q \). The first process sends ‘y’ over channel ‘x’ and continue as process P; parallel to this the second process receives ‘u’ over channel ‘x’ and continue as process Q.

The interaction between two processes can be expressed as \( \underset{xy}{\text{P} } | \underset{x(u)}{Q} \rightarrow P | Q \)

3.1.1 Primitives of \( \pi \)-calculus

Table-I list the key primitives defined by \( \pi \)-calculus.

| Component | Symbol |
|-----------|--------|
| P         | Process |
| P|Q         | Parallel Composition |
| xy        | Output Prefix |
| x(y)      | Input prefix |
| !P        | Replication |
| N         | Restriction |
| 0         | Nil process |

3.1.2 Typing rules of \( \pi \)-calculus

Type system for a programming language helps to reason about the program behaviour. It is also helpful for early finding of bugs. “Type systems for \( \pi \)-calculus are useful both in revealing program errors due to misuse of names and in refining the algebraic theory of the \( \pi \)-calculus”\(^{12}\). Typing rules of a type system is used to prove a valid type judgement. One major purpose of type system is to avoid runtime errors. There is a typing rule for each syntactic form in \( \pi \)-calculus For example T_INP rule

\[
\begin{align*}
\Gamma v: #T, x: T & \vdash P : \Diamond \\
\text{T_INP} \\
\Gamma v(x). P : \Diamond
\end{align*}
\]

This rule specifies the input statement \( v(x) \) is well typed under type environment \( \Gamma \) then \( v \) is of type channel T and \( x \) is of type T and \( P \) is well typed under type environment \( \Gamma \).

3.2 XSB Prolog

Prolog is a general purpose logic programming language where logic is expressed in terms of relation. XSB is a research oriented; commercial-grade logic programming system for UNIX and Windows based platforms. In XSB Prolog evaluation of queries is done according to well-founded semantics through full SLG resolution (tabling with negation). XSB terminates more often than prolog, avoids redundant sub computation and computes the well-founded model of normal logic program\(^{14}\).
4. Proposed system - our approach

In this paper we are proposing a model that uses π-calculus statements that formally capture the complex behavior of the smart environment. We will demonstrate that a smart environment can be effectively modelled in π-calculus and that model can be validated by executing it. A typical example for smart environments is a smart home consisting of a set of interactive services that can be viewed as a collection of different modules like temperature control system, light control system, appliance control system etc. Well typed π-calculus statements can model the different smart environment modules, interaction between different components in each module and their dynamic behavior. Though the example we’ve taken is related to a small part of smart environment, it seems to be a convincing demonstration of the utility of our approach.

To support formal modelling of smart environment using π-calculus, we propose a two phase model which consists of
- Programming smart environment using π-calculus.
- Formal modelling of smart environment using graph.

Our approach is to develop a software application for the execution and observation of the smart environment based on π-calculus. This comprises the ideas for the design of a software application for the execution of π-calculus statements as well as their graphical representation. The graph generated which consists of the components of smart environment and information flow between them can be used to validate the π-calculus based model. Fig. 1 shows a basic framework of our proposed model.

![Frame work for Programming smart environment using π-calculus](image)

4.1 Programming smart environment using π-calculus

The first phase is to represent the smart environment as π-calculus statements that consist of channels and processes. Suppose we want to model a temperature control module of smart environment that include temperature sensors, buffers, processing elements and actuators. The temperature sensors are used to sense the temperature, which is stored in the buffer. A processing element, ‘avg’ calculates the average and sends to ‘threshold’. Based on the threshold value, a command will be send to an actuator say, ‘alarm’. The interactions between different components are done using shared buffers. The above scenario can be expressed using the following π-calculus statement.

\[
\text{TempSensor (sensor).new buf1, newbuf2. Sensor (buf1). avg(buf1,buf2).new buf3.threshold(buf2,buf3). alarm (buf3)}.
\]

Sensors, processing elements and actuators are channels in π-calculus.
4.2 Formal modelling of smart environment (using graph)

In the second phase, the graphical representation of smart environments is generated based on the above \( \pi \)-calculus statement. The different symbols and meanings of the symbol used in the graphical model to represent the components of smart environment are listed in Table II. The information flow between the components is shown by links connecting them. The graphical representation of the above temperature control module is shown in Fig. 2.

| Component                  | Symbol |
|----------------------------|--------|
| Temperature Sensor         | △      |
| Buffer                     |        |
| Processing element/Actuator| ○      |

5. Implementation

The proposed work, programming smart environment using \( \pi \)-calculus is implemented using java and some open Source tools and libraries like ANTLR, XSB prolog, Interprolog and Graphviz. Of the many programming languages available, java seems to be a natural choice as it has multi-threading and communication features built into it. Many open source parser generating tools are available and can be used for parsing the \( \pi \) process definitions from the input file. In this implementation we propose to use ANTLR for generating the parser. ANTLR accepts grammatical language descriptions and generates programs in java that recognize sentences in those languages. As the application is implemented using java, a parser generator producing the output in the same language seems to be more beneficial. Fig. 3 shows the flow diagram of the proposed system.
This application is implemented as two major modules: Language development and Application development

5.1 Language Development

Language development includes generation of a type checker for \( \pi \)-calculus based language, which performs type checking, type inference and sub-typing. Since the input to the application is well typed \( \pi \)-calculus statements, a type system is required to check the well typedness of the \( \pi \)-calculus statements. A type system is a collection of rules that ensures the semantic correctness of each statement of a language. A sound type system for a language avoids runtime errors and thus eliminates the need for runtime error checking. This will further enhance the robustness of the program written in that language. A type-checker prevents type errors during execution by implementing a type system. Type inference is deriving type for a construct based on the type of other constructs. Sub typing is a kind of type polymorphism where a data type, subtype is related to another data type called super type. The relation here is based on the notion of substitution. If S is a subtype of T then S can safely be used wherever T is used. “Subtyping is a pre-order on types that can be thought of as inclusion between the sets of the values of the types. If S is a subtype of T, then a value of type S is also a value of type T. If S is a subtype of T then an expression of type S can always replace an expression of type T.”

In this paper for different \( \pi \)-calculus operations, a set of constructs are defined in a \( \pi \)-calculus based language. The input to the system is a program written in this language. Table 3 lists the \( \pi \)-statements and its corresponding language statements.

| Construct            | \( \pi \)-Statement Keyword | Language Statements |
|----------------------|-----------------------------|---------------------|
| Output Prefix        | xy.P                        | out(x,y).P          |
| Input prefix         | x(y).P                      | in(x,y).P           |
| Parallel Composition | P|Q                          | Par(P,Q)            |

Type systems are typically specified by a set of typing rules. A valid \( \pi \)-calculus statement should conform to its typing rule. In this paper we have implemented the type checker using Prolog, a logic programming language. The typing rules of \( \pi \)-calculus written as clauses in XSB prolog automatically performs type checking and type inference by generating type for each channel. For example, the type rule T-Out can be encoded to XSB prolog as:

\[
\text{hastype}(G, \text{seq(out}(X,Y),P), \text{proc}) :- \\
\text{hastype}(G,X, \text{chan}(T)), \text{hastype}(G, Y, T), \\
\text{hastype}(G, P, \text{proc}).
\]

The different steps involved in language development are as follows:

A program in \( \pi \)-calculus based language is given as input to a parser for syntax analysis. For creating the parser, ANTLR (ANother Tool for Language Recognition) is used. The input to ANTLR is the grammar of the \( \pi \)-calculus based language and output will be the parser generated in java language. Syntactically correct program statements, converted into prolog terms are passed into XSB prolog for type checking and type inference. To connect between java program and XSB prolog we’ve used Interprolog as a bridge.

A Sample code in \( \pi \)-calculus based language is given below.

```java
new y : int [100];
in(x,z); out(z,unit); in(y,12);
```

Intermediate statement generated for type-checking corresponding to the above code is

\[
\text{hastype}(G, \text{seq(new}(n(y), \text{chan}(\text{int})), \text{seq}(\text{in}(n(x),n(z)), \text{seq}(\text{out}(n(z),\text{unit}), \text{seq}(\text{in}(n(y), \text{int}(12), \text{zero})))), \text{proc})
\]

This statement is then type checked based on the typing rules specified in XSB prolog. In the above example only type of channel ’y’ is explicitly given, the type of other channels(x and z) are inferred using encoded typing rules.
5.2 Application development

This is an application in smart environment containing sensors and actuators and its visual representation. The simulation environment developed offers a GUI where the user can select a program based on $\pi$-calculus and the type of sensor to be created in order to start the execution. The implementation includes generation of a parsing component for $\pi$-calculus statements and the creation of a graph representing the smart environment. The parser component reads the input program selected by the user, parses the same and generates the corresponding java program which will then be dynamically loaded, compiled and executed. For this we’ve used reflection.

Sensors are implemented as Sensor Channel, processing elements and actuators are implemented as Processing Element Channel. Each channel has send and receive methods and a Queue data structure to hold the send/receive items. Queue can hold both channel and data. The application is implemented using multithreading, where the GUI will be run as a thread say thread1. The executable code generated will be run as another thread, thread2 and waits for the sensor channel to be created in order to start execution. Thread2 gets blocked if a channel does not exist. When the user creates a channel the thread2 is notified and then it resumes execution. When a new sensor channel is added, at first a global channel will be created. The sensor channel will be sent through this global channel which will then be received by thread2. The GUI also displays the different stages of $\pi$-statement execution. When the statement is being executed, an internal graph is maintained to represent different components. When the statement finishes execution, this graph is traversed and displayed, to show the different components of the smart environment and information flow between them. Graphviz is used for generating visual representation.

Fig 4 shows the visual representation and Fig 5 shows a snapshot of GUI.
6. Conclusion

In case of safety critical systems like smart environments, one major challenge is to provide formal techniques that will enable the efficient design of correct and well-functioning system. In this paper, we have described a basic framework for programming smart environments using well typed π-calculus statements. This method of programming smart environments using π-calculus has proved to show better modelling capability. π-calculus can also be used to verify the smart environments by model checking. Model checking is a formal verification technique which allows for desired behavioural properties of a given system to be verified on the basis of a suitable model of the system through systematic inspection of all states of the model. As future enhancement, formal verification and model checking of smart environments can be done using π calculus.

Acknowledgements

This work derives direction and inspiration from the Chancellor of Amrita University, Sri Mata Amritanandamayi Devi. We thank Dr. P Jayaraj for his guidance and support and Dr. Ramachandra Kaimal, Chairman, Department of Computer Science, Amrita University for his support and valuable feedback.

References

1. Faisal Abouzaid. A mapping from π-calculus into BPEL. Frontiers in artificial intelligence and applications;2006. p 143:235.
2. Cardelli, Gordon. Mobile Ambients. Proceedings of the First international Conference on Foundations of Software Science and Computation Structure ;1998.
3. Sajal K Das, Diane J Cook. Designing and modelling smart environments. Proceedings of the 2006 International Symposium on World of Wireless, Mobile and Multimedia Networks; 2006.p 490–494.
4. Emden Gansner, Eleftherios Koutsofios, Stephen North. Drawing graphs with dot.
5. Jianxiao LIU, Jian WANG, Keqing HE, Xiaoxia LI, Feng LIU. Using π-calculus to model web service interaction. Journal of Computational Information Systems ;2013. p.1759–1767.
6. Robin Milner. Communicating and mobile systems: the π calculus. Cambridge university press;1999.
7. Azza K Nabih, Mostafa M Gomaa, Hossam S Osman, Gamal M Aly. Modeling, simulation, and control of smart homes using petri nets.International Journal of Smart Home. Vol. 5. No. 3; July 2011.
8. Terrence Parr. The Definite ANTLR Reference.
9. Benjamin C Pierce.Type System for Programming Languages.
10. Frank Puhlmann. Why do we actually need the π-calculus for business process management. 9th International Conference on Business Information Systems. volume 85;2006.p. 77–89.
11. Anand Ranganathan, Roy H Campbell. Provably correct pervasive computing environments. Pervasive Computing and Communications., Sixth Annual IEEE International Conference; 2008.p160–169.
12. Davide Sangiorgi. An interpretation of typed objects into typed -calculus.
13. Davide Sangiorgi, David Walker. The π-calculus: A Theory of Mobile Processes. Cambridge university press;2003.
14. Terrance Swift, David S. Warren. The XSB System- Users Manual.
15. Peter Welch. An occam-pi quick reference;1996–2008.
16. Pierce, Turner. Pict A programming language based on the π calculus . Proof, Language, and Interaction; 2000, p.455-494.