POE-Δ integration in a hybrid modelling context

Georgi Markov[1,2] and Jon G. Hall[2] and Lucia Rapanotti[2]

***Abstract*— - first time a formal change management technique has been applied in a formal context - applying formal methods to change management**
JR Tileki TBD

I. INTRODUCTION

- Statement of the Problem
- Type of Research & Research Methodology ??
- Review of Related Literature Look Najun's paper's literature work
- Scope and Limitations
- Significance of the Study

II. BACKGROUND

1) POE-Δ: Problem Oriented Engineering (shortly POE, [1]), defined by the second and third authors, is an engineering framework with an accumulated body of work spanning over a decade, including application and evaluation through a number of real-world engineering case studies. Its underlying design theory concerns the characterization of individual problems and how problems relate and transform to other problems as part of problem solving processes.

POE-Δ [2] is a recent extension to POE which shares and expands on a number of POE's characteristics, including elements of its semantics; its graphical notations; and its underlying process pattern [1]. However, while POE deals with 'greenfield' development, POE-Δ deals with change, or 'brownfield,' problems which are solved not solely by the design of a new artifact, but by a change of, and addition to, existing artifacts within a target context (a system, product, process, etc).

A thorough presentation of POE and POE-Δ is beyond the scope of this paper, but can be found in [2]. Here, we very briefly recall some basic definitions and concepts important in the context of this paper.

In POE a problem is "a stakeholder's recognized need in context." For stakeholder G, with recognized need $\mathsf{N_G}$ in real world context $\mathsf{E_G}$, we defined their problem to be the pair:

$$(\mathsf{E_G}, \mathsf{N_G})$$

$\mathsf{E_G}$ and $\mathsf{N_G}$ are to be understood only as place holders, as G's initial conceptualization of their problem may have neither solution nor sense. Irrespective of sense or solution existence, G's wish becomes a challenge to designer D to make sense of G's problem by finding an agreed environment E and need N, leading to D's problem

$$\mathsf{E(S)} \Vdash_{\mathsf{G}} \mathsf{N}$$

which reads "Find S which, when installed in E, meets N to the satisfaction of G."

D's challenge consists of all the solving problems activities that lead to the solution of G's problem. Someway through problem solving we encounter D's variously detailed E, N and S and form a judgement as to whether a problem has been solved. We do this by creating a solution for it through a sequence of judgement-preserving transformations, i.e., transformations that the relevant stakeholders would agree preserved solvability, that move a problem to known solved problems. Thus, a problem is solved if and only if it can be transformed to known solved problems. As part of the transformation sequence, a solution to the problem is created.

In POE-Δ, we begin from the same place as POE: we suppose that change problem owner G recognizes a need in the real world and wishes that need to be satisfied. From G's perspective, then, a problem P is a pair, consisting of a real world context $\mathsf{E_G}$ and a need $\mathsf{N_G}$.

Irrespective of sense or solution existence, we will assume that G's wish becomes a challenge to a change engineer D to make sense of G's change problem $(\mathsf{E_G}, \mathsf{N_G})$ and to solve it. D's challenge thus consists of (cf. [3]):

CPS1. creating their own view, (E, N), of the G's change problem $(\mathsf{E_G}, \mathsf{N_G})$, and identifying domains that must remain unchanged, those that can change;
CPS2. receiving validation from G that (E, N) is properly representative;
CPS3. either, identifying a new environment F satisfying the change need which
  i. preserves those parts of E that should remain unchanged,
  ii. identifies any changes necessary to domains that can change, and
  iii. identifies any additional domains necessary to effect the change;
  or reporting that the change isn't possible.
CPS4. receiving validation from G that F meets the agreed recognised changed need N;
CPS5. migrating from E to F.

Like POE [3], even if expressed linearly as bullet points, the challenge facing D may be iterative and highly non-linear.

Although $\mathsf{E_G}$ and $\mathsf{N_G}$ are expressed in stakeholder language, here we assume that the developer has access to a full formal tool box and is able to express both E and N in

*This work was not supported by any organization
[1]Siemens Corporate Technology, Princeton, NJ 08540, USA georgi.markov at siemens.com
[2] Open University, Milton Keynes, MK7 6BJ, United Kingdom {georgi.markov, jon.hall, lucia.rapanotti} at open.ac.uk

a suitable formal language. For this paper, we assume that language is Hybrid CSP [4].

2) The Mechanism Description Language and the Phenomenal Model: How do we integrate divere disciptions on a semantic level?

- Phenomenal Basis for Hybrid Modelling
- Naijun Zhan - From HCSP to Simulink paper
- etc.

3) HCSP: Hybrid CSP is an extension of Hoare's original CSP calculus [5], [6] to provide the necessary constructs for modeling continuous, communicating and real-time behavior as exhibited by many hybrid systems (systems whose behavior combines continuous and discrete dynamic), including but not limited to:

- Differential equations to model continuous evolution, including the ability to preempt the evolution by events.
- Different types of events and interruptions to enable modeling interactions between continuous evolutions and discrete jumps.
- Event communication to enable interaction among processes.

While a thorough description of HCSP vocabulary can be found in Chaochen et al's original work [4], here we discuss a few of the HCSP constructs on a small example, which combines a number of primitives - input, output, assignment) and some composite operators - sequential and parallel process execution into the following more complex expression:

$$ch!5||(ch?\mathsf{x};\mathsf{y} := \mathsf{x})$$

We break the expression into its constituent HCSP simple terms:

- Channel Output: $ch!5$
- Channel Input: $ch?\mathsf{x}$
- Value assignment: $\mathsf{y} := \mathsf{x}$

Each such HCSP term's semantic is briefly discussed in the following and it is shown how such HCSP simple terms can be represented and how using the Phenomenal Model defined on top of the Mechanism Description Language presented in II-.2 we can integrate these HCSP terms into our POE-Δ calculus.

a) The channel output: ch!5: In [4], the semantics of $ch!5$ are defined as:

$$[\![ch!\mathsf{x}]\!]\mathsf{C} \triangleq \begin{array}{l} wait(\alpha, ch!) \wedge keep \\ \vee \\ (wait(\alpha, ch!) \wedge keep) \frown syn(\alpha, ch!, \mathsf{x}) \frown \mathsf{C} \end{array}$$

The following two diagrams 1 provide a visual representation of these semantics:

Each diagram presents one of the two cases which satisfy the $ch!5$ semantics. The top diagram depicts the case where $ch!$ it is willing to communicate (returns $true$ - tt.), while the $ch?$ term returns $false$- ff., meaning it is unwilling to communicate. In this case the behavior of $ch!5$ is characterized by $wait \wedge keep$ and the value on the channel ch is irrelevant.

The bottom diagram, shows the second case, which satisfies the $ch!5$ semantics. It subdivides the $[\mathsf{c}, \mathsf{d}]$ interval in 3 phases:

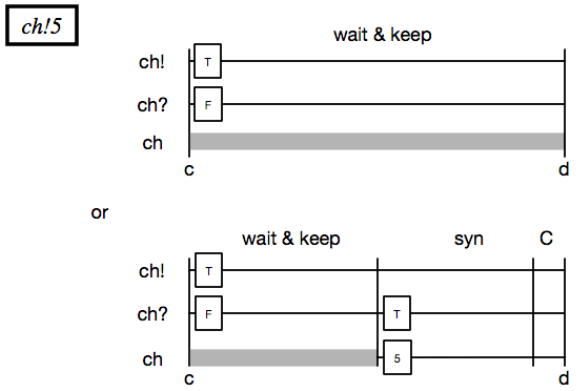

Fig. 1. Graphical depiction of the HCSP channel output semantics

- The first interval segment is analogous to the first diagram - $wait \wedge keep$. In the second segment, the value of $ch?$ turns to true - tt., meaning that now both - $ch!$ and $ch?$ are willing to communicate, so we move to the syn-phase, where the value on the channel ch is send to x.
- The last phase of the interval describes the continuation sequence - C.

The POE/POE-Δ representation of the ch!5 term is the ch!5 HCSP domain with the following semantic: $[\![ch!5]\!]\lceil$.

b) The channel input: ch?x: The next diagram pair 2 depicts the second term in our example - $ch?$x:

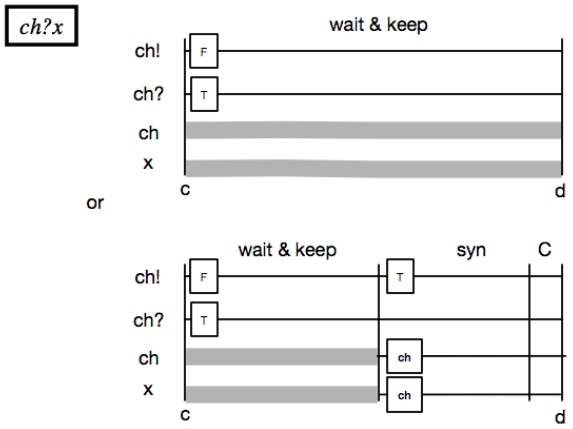

Fig. 2. Graphical depiction of the HCSP channel input semantics

Analogous to 1, this diagram pair discusses the two cases that satisfy the $ch?$x term. In the top diagram, $ch!$ evaluates to tt., while $ch?$ to ff.. In this case the values of ch, and x are irrelevant and the behaviour of the term is simply $wait \wedge keep$. In a equivalent manner in the bottom diagram the interval is subdivided in three phases:

- In the first phase, $ch?$ is $false$, $ch!$ evaluates to $true$, so the behaviour is $wait \wedge keep$.
- Once $ch?$ turns to $true$, we move to the syn phase where value on ch is moved to x.
- The last phase of the behaviour describes, again, the continuation sequence.

In POE/POE-Δ, the ch?x HCSP term is represented with the ch?x domain with the semantics: $[\![ch?x]\!]\lceil\rceil$.

c) Value Assignment: y:=x: The last primitive term, the assignment operation, which is visualized in Figure II-.3.c simply sets the value of y to the values of x in the time interval - [c, d]

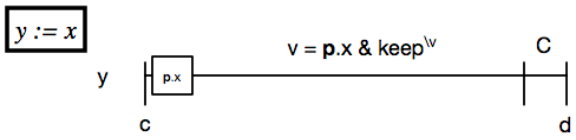

Fig. 3. Graphical depiction of the Assignment semantics

The POE/POE-Δ representation of the Value Assignment term is y := x domain with the semantics: $[\![y := x]\!]\lceil\rceil$.

d) Building complex terms from primitives: For scaling up to real live systems, the HCSP primitive terms can be combined into complex expressions using compositional operators like sequential and parallel composition, as this is the case in our example - $ch!5||(ch?x; y := x)$. First, let's look at the sequential operator. In HCSP the sequential operator is defined as follows:

$$[\![P; Q]\!]C \triangleq [\![P]\!]([\![Q]\!]C)$$

The following Figure 4 depicts the semantic of the sequence operator. The time interval [c, d] is split into two subintervals: [c, e] which is assigned to P and [e, d] which is assigned to Q, where $c < e < d$. The satisfiability, and thus the result of the expression depends on the existence of a value $'v'$, for which the first HCSP term returns true in it's dedicated subinterval. If such a value exists the control flow is passed to the second term and its return value is then returned as the return value of the composite expression.

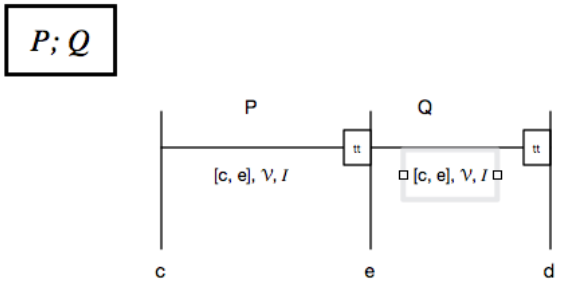

Fig. 4. Graphical depiction of the sequential operator semantics

The other operator used in the example is the parallel composition operator, whose semantic is defined as follows:

$$\begin{aligned}[\![P||Q]\!] \triangleq \quad & [\![P]\!]skip \wedge [\![Q]\!]skip \\ & \vee \\ & [\![P]\!]skip \wedge [\![Q]\!]stop^{Q} \\ & \vee \\ & [\![P]\!]stop^{P} \wedge [\![Q]\!]skip\end{aligned}$$

Figure 6 visualises that semantic of the parallel operator, which given a time interval [c, d] gives control to both processes P, and Q.

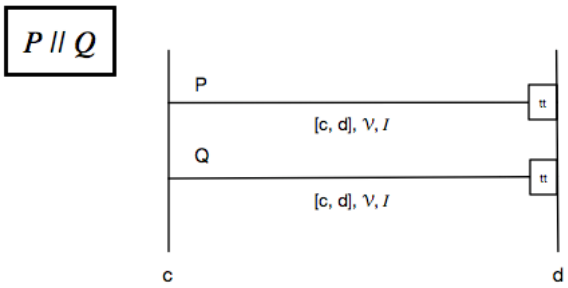

Fig. 5. Graphical depiction of the sequential operator semantics

The POE/POE-Δ representation of the sequencial and parallel composition are the sequencial, respectively the parallel HCSP domains: ||, ; .

e) Application of HCSP in POE/POE-Δ: As shown in the previous section we use the notion of HCSP unit domains,to represent each HCSP primitive as a POE/POE-Δ unit domain and plot it in a problem diagram. Additionally, we use the POE/POE-Δ idea of shared phenomena, represented as arc annotations to capture information about shared state as well to coordinate the flow of execution. Figure **??** shows a problem diagram representing our running example from the previous section.

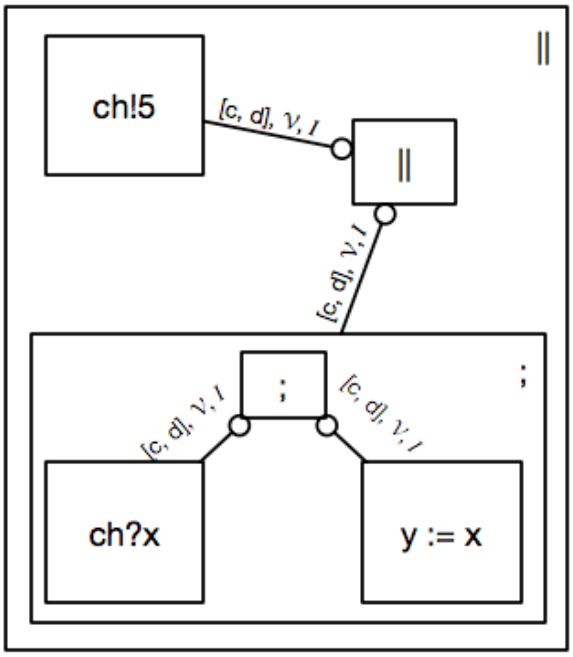

Fig. 6. Graphical representation of the $ch!5||(ch?x; y := x)$ example in POE

In the context of HCSP, the original POE/POE-Δ problem definition:

$$E(S) \vdash_G N$$

which reads "Find S which, when installed in E, meets N to the satisfaction of G." can be reinterpreted as:

$$\forall v \in V, i \in Intv(E)L([E](S)), v, i = tt.$$

meaning that for all initial assignments V, and for all Intervals 'i' which are allowed for E, the interpretation of ([E](S)) in Interval 'i' and with valuation 'v' which evaluate to true satisfy the need N.

With this reinterpretation, we can use the higher level reasoning capabilities offered by POE/POE-Δ, while remaining fully compatible with the precise and executable semantics of HCSP. The following example motivated by the Inventory Management System from Chaochen's original paper [4] illustrates this idea. Additionally it builds up on the previously discussed work of Mingshuai Chen, et al [7] to allow us to take advantage of HCSP's executable semantics during the validation steps of the POE-Δ process.

III. EXAMPLE

a) The Kettle Problem: In the following we introduce a very simple example of a heating kettle with initially only a heating actuator and in a second version also a temperature reading sensor, illustrative of a plant/controller problem as the one presented in Chaochen et al.'s original paper [4] and depicted graphically in the following Figure 8.

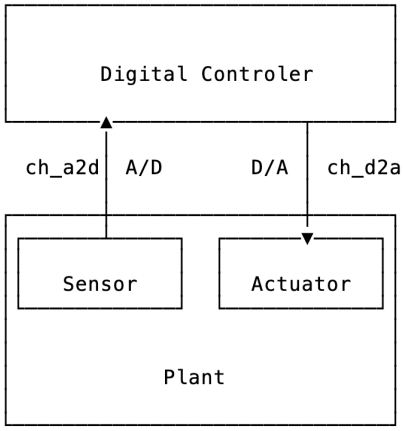

Fig. 7. The Interfaced Plant problem from [4]

On the basis of this example we demonstrate how Chaochen's HCSP calculus and our POE-Δ change calculus can be integrated in a hybrid modeling approach by reinterpreting HCSP in terms of real-world phenomena and domains using our previously published work on *Mechanism Description Languages* [8].

Such a deep semantic integration of the two description systems results in a more formal change framework with executable semantics. Using a Simulink implementation (based on the work of Chen at al. [7] and kindly made available by Dr. Naijun Zhan[1]) of these semantics is then applied to analyze and validate a change scenarios on the presented Kettle example - one where the change intervention involves the addition of an additional sensors as a result of the change ramifications to the rest of the system.

Example 1 (The original Kettle v1.0 model): Kettle 'v1.0' is the original version of our old-fashioned heating kettle, which the producing company (and change problem owner) has decided to overhaul and make ready for the IoT world. It is a real-world located device that heats water to approx. boiling point for brewing tea and coffee. It has only a heating actuator, but no temperature reading sensor. Additionally, the Kettle module is controlled by a software logic, which ensures that once turned on, the kettle heats up water at a rate of approx. 1 °C/sec for the duration of $60sec.$, after which it automatically turns off. It is described by the following equations:

$$HeaterModule :: \langle\ \dot{e} \quad = \quad 0\ \rangle$$

The $HeaterModule$ has only an digital-to-analogue convertor:

$$KettleModule :: (HeaterModule \triangleright ch_{\mathtt{d2a}}?\mathtt{e})$$

and the software controller is described by the following logic:

$$\begin{array}{lll} & ch_{\mathtt{d2a}}!off; & \%\ \text{Initial state is off} \\ & \mu\mathsf{X}.(\mathtt{Stop} \trianglerighteq & \%\ \text{wait for boil request} \\ & \quad (boilRequest?\mathtt{x}; & \\ & \quad\ ch_{\mathtt{d2a}}!on; & \%\ \text{turn heater element on} \\ SWProgram1.0 :: & \quad\ \mathtt{Wait}_{60}; & \%\ \text{for 60 seconds} \\ & \quad\ ch_{\mathtt{d2a}}!off & \%\ \text{then turn if off;} \\ & \quad\ \mathsf{X} & \%\ \text{and repeat} \\ & \quad) & \\ &) & \end{array}$$

The whole $Kettle$ is then

$$Kettle :: HeaterModule \parallel Kettle :: SWProgram1.0 \models KNeed$$

and can be represented in a POE-Δ problem diagram as follows:

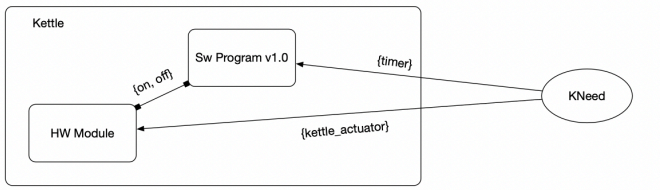

Fig. 8. A Problem Diagram of the initial Kettle

In this original version, the original KNeed is simply:

- - KNeed: continuous operation (t is temperature, e is constant on/off)

Using our semantic function in combination with the HCSP-Simulink building blocks defined in [7], we are able to derive an executable Simulink model of the kettle's behaviour and use it during validation. Figure 9 shows an aggregated view of the Simulink diagrams describing the kettle:

As expected, the simulated kettle behavior demonstrates the following explicit properties:

- Water temperature before activating the heating module is at constant ambient temperature of 21°C, as modeled

[1] http://lcs.ios.ac.cn/~znj/

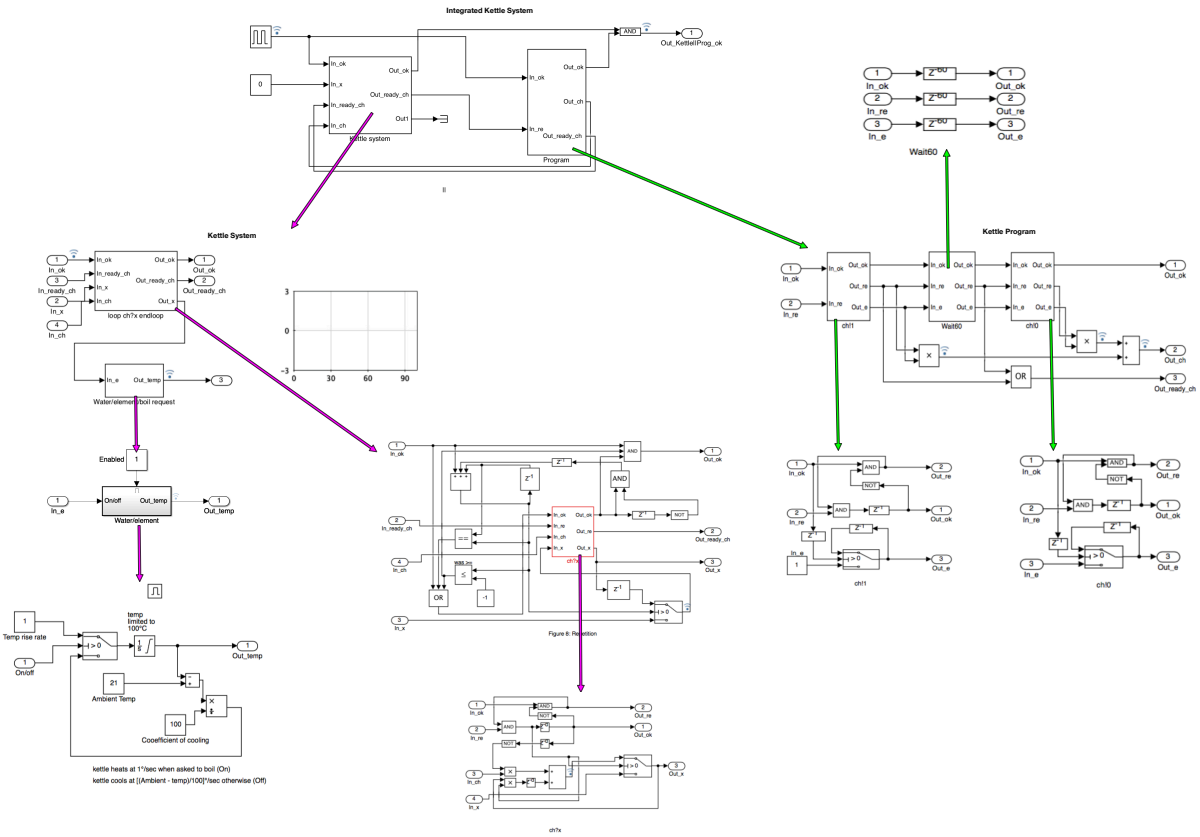

Fig. 9. A Simulink model of the kettle behavior

in the water element diagram, which can be found on the bottom right corner of the aggregated Simulink Figure.

- Once switched on, the kettle heats the water at $1°C/sec$ for the duration of 60sec., which in thE normal case does not allow the kettle to heat up to ¿100°C.
- Once switched off the water temperature cools off at a rate of $[(Ambient\,Temp - Temp)/100]°C$

A graph of this behavior is show in the following Figure 10

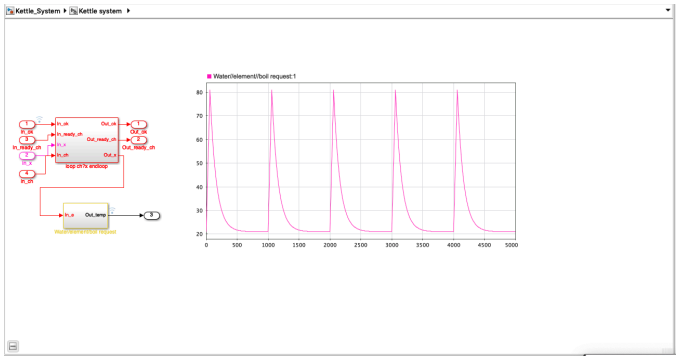

Fig. 10. A Graph of the Kettle behavior

b) The Kettle change problem: In the following, we will analyze a change introduced by a new IoT-Kettle need, its impact on both designs and how POE-Δ facilitates the analysis and change process. The additional need is:

- $KNeed_{New}$ - KN_n1: WiFi/Bluetooth Connectivity for remote management (e.g. through iPhone app)

So the change problem in POE-Δ is now:

$$Kettle :: HeaterModule[Kettle :: SWProgram1.0]\Delta\mathsf{F}(WiFi/Bluetooth1$$

and our task to to find F, which is a placeholder for the changes required to the Kettle.

c) Step 1: Change Localization: The initial step in POE-Δ - $CPS1$, as explained in section II-.1, involves capturing an initial view of the change problem, and identifying domains that must remain unchanged and such that will change.

In real life problems, this is usually a multi-step process, which is facilitated by the POE/POE-Δ process pattern, and involves analyzing the changes to the needs / requirements, and their impact on the current design. While a more detailed example of such an analysis and the tradeoffs which it might require can be found in [2], in our, relatively simple example, this multi-step process is summarized in a single step as follows:

During the initial analysis of the problem, the architect identifies that:

- the high-level system architecture will need to change to accommodate the WiFi, Bluetooth hardware modules
- the Sw. Program v1.0 will need to change to accommodate the control logic necessary to address the new requirement

The following POE-Δ diagram graphically depicts that change:

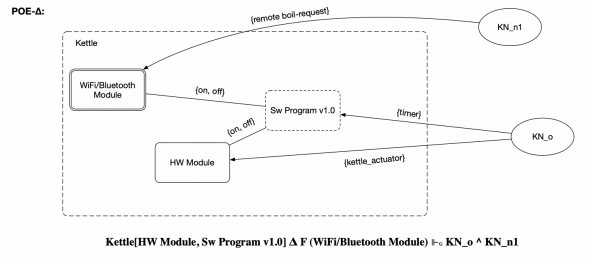

Fig. 11. Initial Change analysis in POE-Δ

Using either of these low-level executable models (HCSP or Simulink), this initial change design can be formally proven or verified through the use of simulations. In our case this step reveals a serious flow in the design – the introduction of the networking component - *WiFi/Bluetooth Module*, which can be hacked and exploited, we introduce a number of potential safety and security concerns, each of which could compromise the kettle's safe operation. One such issue is illustrated in the following Figure, which shows how a malicious attack can exploit a design flow in the original kettle when the idle times between runs are too short for the water to cool down sufficiently.

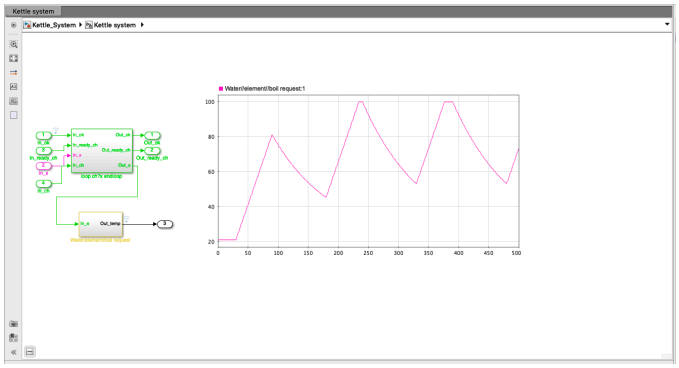

Fig. 12. A Graph of a potentially hazardous behavior of the kettle

This issue introduces additional requirements to the new version of the Kettle system:

- KN_n1: WiFi/Bluetooth Connectivity for remote management (e.g. through iPhone app), which however due to safety / security concerns (e.g. hacker can attack WiFi enabled kettle and make it burn the house), introduces additional requirements:
  - KN_n2: Guaranteed Safety (t shall not raise $>$ 90°Celsius to prevent overheating)
  - KN_n3: Self-learning AI-based security component aimed at determining usual behavior of the devices, and taking countermeasures when behavior starts looking abnormal, e.g. turn off kettle during attacks

d) Step 1a: Change Localization: At this point we can come back to our high-level POE-Δ change analysis language, and analyze the new change problem:

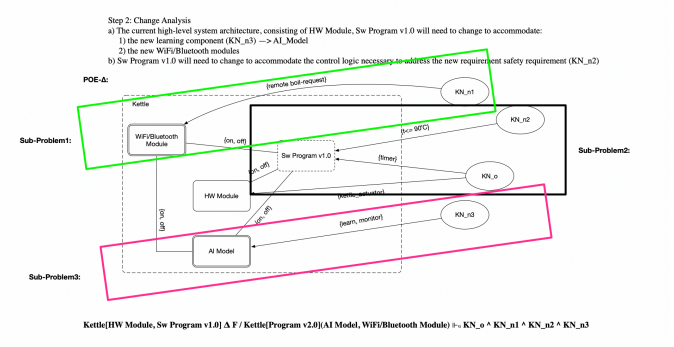

Fig. 13. High-level Change Analysis

In the reminder of this example, we will assume that the three Sub-Problems are cleanly separable and we choose to focus on Sub-Problem2, in order to illustrate the benefits of our approach. The benefits of POE-Δ in cases where such clean separation between sub-problems is not possible has been described in [2].

During this step the System Architect identifies that the current $ch_{\mathsf{d2a}}!on; Wait60; ch_{\mathsf{d2a}}!off$ control logic is where the change will need to be made. In particular the current logic does not include any safety instrumentalization - once on, the KettleModule remains on for a fix amount of time (60 sec) and does not consider any feedback from the temperature sensor regarding the current temperature of the heating module. This in a defective, or hacked unit could result in a safety hazard. The change localization is shown graphically in the following figure, which uses a new graphical notation – combining elements of our high-level POE-Δ and the low-level HSCP.

$$Kettle :: HeaterModule[Kettle :: SWProgram1.0]\Delta Program[Wait60]()$$

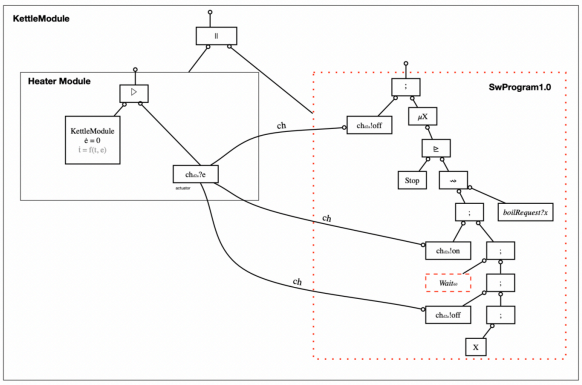

Fig. 14. The change localisation

e) Step 2: Design for change through the application of the POEΔ Substitution Pattern: Once the change has been localized, POE-Δ guides us also through the changes, including the design and introduction of new components, the removal and change of existing component, and the analysis of the potential ripple effects such changes might

have on the rest of the system, for example due to interface incompatibilities.

In order to satisfy the new requirement, the control logic needs to be able to consider feedback from a temperature sensor in its decision on whether to keep heating up or turning off the heating actuator. As we identified in the previous step, this means that the current module $Wait_{60}$ will need to be replaced with a new module:

$$Sub[Wait_{60}/SefetyPatch]$$

where

$$SafetyPatch := \mu(Wait_1, ch_{\mathsf{a2d}}?\mathsf{t}; \Upsilon if\, \mathsf{t} < 90)$$

so that:

$$Kettle :: HeaterModule[Kettle :: SWProgram1.0]\Delta Program[Wait60 \triangleright \mu(Wait_1, ch_{\mathsf{a2d}}?\mathsf{t}; \Upsilon if\, \mathsf{t} < 90]() \models KNeed+$$

As such the designed new replacement module introduces the need to be able to receive feedback from the $Kettle :: HeaterModule$ about its current temperature through a sensor which is currently not part of the Kettle's HW design. In such a way the change introduced to one part of the system ($Kettle :: SWProgram1.0$) has a ripple effect and requires the introduction of a cascading change to another part of the system - in this case a new sensor for temperature reading - $ch_{\mathsf{a2d}}!\mathsf{t}$ in the $Kettle :: HeaterModule$. This is indicated in the Figure 15 by the color coding.

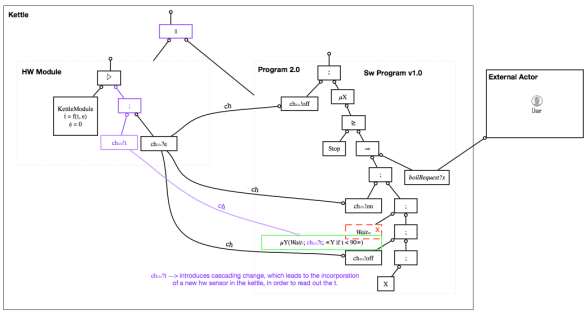

Fig. 15. The change to $Kettle$ 'B' and its implications

Such cascading changes are a common phenomenon in change problem of real-life complexity. The ability of $POE-\Delta$ to explore the interdependent nature of change sub-problems which extend beyond the scope of the system, but also in the environment and to support the change engineer in the solutions co-design is one of its main advantages compared to classical problem decomposition approach of divide and conquer.

f) Step 3: Validating the change design: For the purpose of validation, we use again our previously introduced executable model of the kettle behavior, adapt it according to the just designed change and use its simulation ability during validation.

To identify the building blocks, which need changing in the Simulink model easier, we used Stateflow's build-in report functionality which extends Simulink's default Print Details report. It provides the ability to generate HTML reports of the entire Simulink mode. The generated raw report represents the different model elements and connections in a table form (see Figure 16) which is very helpful in quickly identifying the necessary changes. To additionally support the usability of this approach, we applied some transformation in order to further compact and summarize the information. The resulting report, which can be seen in Figure 17 was embedded in our POE-Δ framework in order to localize and perform the changes in the model necessary to make the Kettle Simulation conform with the changed Kettle from the previous step.

A graph of the behavior of the changed Kattle design according to the new Simulink model is show in the following Figure 18

shouldn't the kettle keep the heat on until we reach 90 degree Centigrade?

The success of the validation step, in particular in problems of non-trivial complexity is not guaranteed. Often the validation fails and the change analysis steps needs to re-trace. Of particular difficulty for the test designer is the necessity to understand the runtime impact the POE-Δ changes might introduce. The existence of an executable Simulink model derived through the integration of POE-Δ and HCSP was a significant step towards addressing this problem, since now the runtime behavior of the system can simulated and directly observed.

After a successful validation,. he last remaining step for satisfying the change requirement involves the actual change realization, which comprises two separate phases - the decommissioning of the old system and the commissioning of the new system. This

IV. CONCLUSION

- Concluding Statement Through the proposed integration of POE-Δ - the high-level change design calculus and a low-level formalism with executable semantics – in this case HCSP, we can enable a seamless transition between different abstraction levels in the context of the overall change process. The resulting approach provides a systematic way for representing and analyzing technical changes, pinpointing the exact intervention necessary in the system, but also showing its potential impact to the rest of the system as well as to the incorporating environment at a high level, while at the same time enabling the automatic validation of the high-level changes by means of a semantic mapping of POE-Δ to an executable formalism. Using a Simulink implementation of these semantics is then applied to analyze and validate a change scenarios on the presented Kettle example - one where the change intervention involves the discovery of new requirements and the addition of an additional sensors as a result of the change ramifications to the rest of the system.
- Next Steps

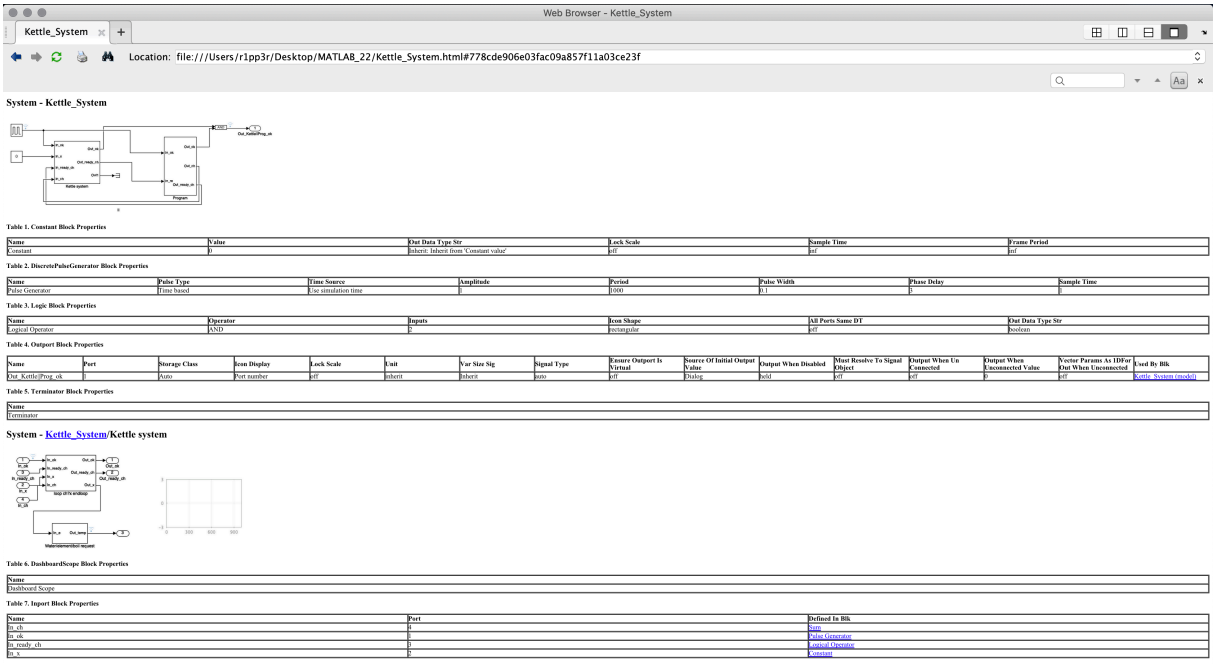

Fig. 16. Excerpt of the Raw HTML Simulink Report

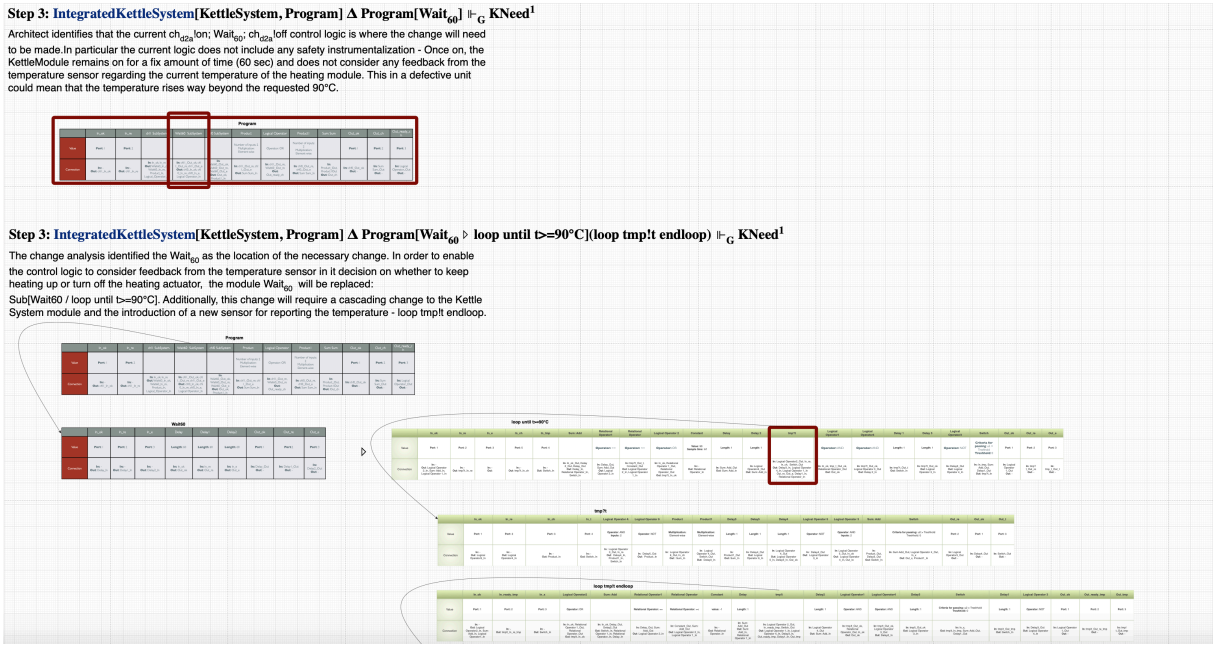

Fig. 17. Excerpt of the application of POE-Δ on the Modified Simulink Report

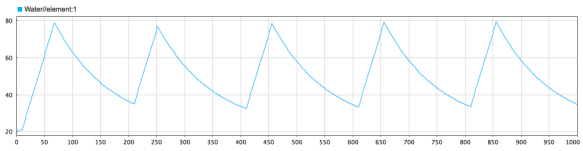

Fig. 18. The simulated behavior of the changed kettle.

APPENDIX

Diagrams and larger graphics??

ACKNOWLEDGEMENT

Naijun Zhan - http://lcs.ios.ac.cn/ znj/ for sharing the original implementation

REFERENCES

[1] J. G. Hall and L. Rapanotti, "Assurance-driven design in problem oriented engineering," *International Journal on Advances in Systems and Measurements*, vol. 2, no. 1, pp. 26–31, 2009.

[2] G. Markov, J. G. Hall, and L. Rapanotti, "Poe-Δ,"

[3] J. G. Hall and L. Rapanotti, "A design theory for software engineering," tech. rep., Technical Report TR2016/01, Department of Computing and Communications, The Open University, Walton Hall, Milton Keynes, MK7 6AA, 2016.
[4] Z. Chaochen, W. Ji, and A. P. Ravn, "A formal description of hybrid systems," in *Hybrid Systems III*, pp. 511–530, Springer, 1996.
[5] C. A. R. Hoare, "Communicating sequential processes," *Communications of the ACM*, vol. 21, no. 8, pp. 666–677, 1978.
[6] C. A. R. Hoare *et al.*, *Communicating sequential processes*, vol. 178. Prentice-hall Englewood Cliffs, 1985.
[7] M. Chen, A. P. Ravn, S. Wang, M. Yang, and N. Zhan, "A two-way path between formal and informal design of embedded systems," in *International Symposium on Unifying Theories of Programming*, pp. 65–92, Springer, 2016.
[8] J. G. Hall, L. Rapanotti, and G. Markov, "A phenomenal basis for hybrid modelling," in *Information Reuse and Integration (IRI), 2017 IEEE International Conference on*, pp. 291–297, IEEE, 2017.