CEUR Copyright Notice

Published in: ***Proceedings of the 3rd International Workshop on Interplay of Model-Driven and Component-Based Software Engineering co-located with ACM/IEEE 19th International Conference on Model Driven Engineering Languages & Systems (MoDELS 2016),*** October 2016

"Feature-Oriented Modelling in BIP: A Case Study"

Cite as:

Cecylia Bocovich, Joanne M. Atlee, "Feature-Oriented Modelling in BIP: A Case Study," In *Proceedings of the 3rd International Workshop on Interplay of Model-Driven and Component-Based Software Engineering (ModComp'16),* October 2016, pp. 6-11.

BibTex:

```
@inproceedings{DBLP:conf/models/BocovichA16,
  author    = {Cecylia Bocovich and Joanne M. Atlee},
  title     = {Feature-Oriented Modelling in {BIP:} {A} Case Study},
  booktitle = {Proceedings of the 3rd International Workshop on Interplay of
Model-Driven and Component-Based Software Engineering co-located with
{ACM/IEEE} 19th International Conference on Model Driven Engineering
Languages {\&} Systems (MoDELS 2016), Saint-Malo, France, October 2nd, 2016},
  pages     = {6--11},
  year      = {2016}
}
```

URN: http://ceur-ws.org/Vol-1723/

Feature-Oriented Modelling in BIP: A Case Study

Cecylia Bocovich Joanne Atlee
University of Waterloo
Email: {cbocovic, jmatlee}@uwaterloo.ca

***Abstract*—In this paper, we investigate the usage of Behaviour-Interaction-Priority version 2 (BIP2), a component-based modelling framework, for specifying feature-oriented systems. We evaluate BIP2 in the context of the Feature Interaction Problem and quantify the amount of work needed to add features to an existing system (i.e., in terms of rework to existing features, and work to identify and specify interactions). We present the results of a case study on a telephony system with five optional features where we found that the amount of work depends heavily on how features are interconnected. We identify three different design methodologies for interconnecting features, and propose one that reduces the amount of work and rework needed to add new features to an existing system.**

I. Introduction

In software engineering, an increasingly popular strategy to decompose a complex system into smaller subproblems is to perform **feature-based decomposition**, which is a type of functional decomposition of the system. A **feature** is a unit of functionality that can be developed and evolved independently. However, the composition of separately designed features to produce a final product often leads to unexpected or undesirable behaviours. A **feature interaction (FI)** occurs whenever the presence of one feature alters the behaviour of another. For example, a user may subscribe to a telephony feature that automatically forwards her calls to another number; she may also subscribe to a second feature that screens calls against a list of blocked numbers. If each feature is specified and developed without knowledge or consideration of the other feature, the outcome is not clear when both are activated in the same scenario. A call could be screened before it is forwarded, or it could instead be screened against the list of blocked numbers at the forwarding destination.

To be safe, a developer must consider how a new feature might interact with existing features. To be thorough, all combinations of existing features need to be considered. As the number of features grows, the number of feature combinations that must be analyzed for possible interactions grow exponentially — until the work of integrating a new feature is dominated by the analysis and resolution of feature interactions. In systems with high variability, the **Feature Interaction Problem**, the task of analyzing every possible combination of composed features and resolving any discovered feature interactions, becomes intractable with existing methods [1].

Many techniques and tools have been developed to minimize the work of the developer in discovering and resolving feature interactions [2]. One such strategy is the use of specialized modelling languages for the design and verification of composed systems. **Behaviour-Interaction-Priority version 2 (BIP2)** [3], [4] is a framework for the design of component-based systems. BIP2 allows the designer to decompose a complex system into a collection of interconnected components.

Given that the BIP2 formalism is designed to support component-based modularity, and given that BIP2 has explicit language constructs for specifying how feature combinations ought to synchronize and how conflicts and nondeterminism ought to be resolved, we investigated how to use BIP2 to address the Feature Interaction Problem. We performed a case study in which we used BIP2 to model a telephony system with five features. We aimed to answer the following questions in our investigation: (1) Is it possible to model features independently and integrate them into the system without changing existing features? (2) How much work (and rework) is required to integrate a new feature into an existing system model? (3) How much work is required to specify interactions among features, and what is the overall complexity of the resulting system model?

Answers to these questions depend heavily on the design methodology used to define component interfaces and to interconnect components. We identify three distinct design methodologies for composing features, and we evaluate the amount of developer work that is needed to integrate new features and resolve feature interactions in each approach. An interesting side effect of this work is that we have shown how BIP2 – whose strength is in modelling components that are *designed to know about each other and to work together* – can be used to model components that *do **not know** about each other* and to compose them so that they can work together.

II. Overview of BIP

Behaviour-Interaction-Priority (BIP) is a component-based language for modelling complex systems [3]. In BIP, the behaviour of a system is modelled as a collection of individual components, each of which is responsible for a subset of the system's behaviour. As the name suggests, BIP provides three layers of specification to the model: the Behaviour of system components, the Interactions [1] between these components, and the Priorities between multiple possible execution paths. In this paper, we use the second iteration of this framework, BIP2 [4], and will refer to this version from this point forward.

[1]Given how the term *interaction* is overloaded, we use the acronym **FI** to refer to a traditional *feature interaction* (any difference in feature behaviour, intended or not, due to the presence of other features). We reserve the qualified term **interaction** to refer to a *BIP2 interaction* (an explicitly specified communication and synchronization among connected components).

A. The Behaviour Layer

Each **component** in a BIP2 model defines a subset of a system's overall functionality. In this paper, our system consists of a base component that provides basic call-processing functionality (i.e., on-demand voice connections between two users), a set of optional feature components that extend or override this functionality, and a component that represents the system's environment (i.e., telephone users).

The most basic BIP2 component is an **atom**. The internal operation of each atom is modelled as a Petri net. An atom's current state is represented by the set of currently occupied places and the values of the atom's variables. Transitions between places in the net update the atom's variables and the set of occupied places. A transition from a set of previously occupied places to a set of newly occupied places may be optionally labelled with a guard, an update function, and a port. A guard is a predicate over the atom's variables, and a transition is enabled and executed only if the system state satisfies the guard. After transitioning, the variables are updated as dictated by the update function. Ports trigger transitions in synchronization with other components, and are used in the specification of the interaction layer. Ports restrict transitions similar to guards; a transition labelled by a port relies on an interaction with another component to execute.

B. The Interaction Layer

Components interface with each other through ports that are linked together by **connectors**. A connector links at most one port from each of the two or more components it connects: the effect is to synchronize the transitions in each of the connected components that are labelled with the linked ports. The ports in a connector may be either **triggering ports** (i.e., senders) or **synchronizing ports** (i.e., receivers). When a transition labelled with a sender (denoted by a primed port name, e.g., busy') is enabled, a synchronized execution step that involves a subset of the enabled receiving transitions in the connected components will execute. The subset of transitions that execute is determined by the guards and the priority ordering of the connector's **interactions**.

Each interaction in a connector consists of a triggering port(s) and some subset of the connector's synchronizing ports. Interactions may be labelled with guard and transfer functions in the same manner as component transitions, restricting which of the components will participate in the synchronized step. The variables in these functions are the data variables exported by the components' ports. Upon execution, the interaction's transfer function updates the variables in participant atoms, allowing components to exchange information.

For example, in a telephony model, the connectors between the basic-call service components of multiple users define the ways in which the services may interact throughout the process of a call. Likewise, the connectors between a user component and its basic-call component define how a user interacts with her own call service.

C. Priorities

To combat nondeterminism and enforce scheduling policies, BIP2 provides **priorities** as a means to choose between multiple enabled execution paths. Nondeterminism arises when there are multiple simultaneously enabled interactions, each leading to a different overall system state. Normally, if there is more than one connector with an enabled interaction, there are no guarantees about which interaction will execute. We can control the outcome by specifying priorities in one of two ways: (1) at the component level by specifying that port p_1 has a higher priority than port p_2 with $p_1 > p_2$, or (2) at the interaction level by specifying that interactions in the connector C_1 have priority over interactions in the connector C_2 with the priority $C_1 : * > C_2 : *$.

The simplest way to resolve all nondeterminism is to define a complete ordering on the transitions that lead from each state. Our basic-call service atom requires a total of 26 priorities to resolve conflicts from simultaneously enabled interactions and avoid inconsistent states. Priorities play a large role in the resolution of feature interactions.

III. Telephony Case Study

We conducted a case study on a telephony system to assess the extent to which BIP2 combats the Feature Interaction Problem. In this section, we outline the basic structure of our telephony system, the features involved, and the criteria we used to evaluate the design methodologies we developed.

A feature-oriented BIP2 telephony model consists of three parts: (1) a **basic-call service** (modelled as an atomic component), (2) a set of optional **features** to which a user may subscribe that extend or modify the functionality of the basic-call service (each of which is modelled as an atomic component), and (3) the **user** (modelled as an atomic component).

Each user's basic-call service (BCS) allows that user to place and receive calls. The places in the BCS component, together with its variables, reflect the possible states of an outgoing or incoming call. The ports of the component reflect the ways in which users and features may interact with or extend the functionality of the BCS (e.g., taking the phone off the hook, or dialing a number), and the ways in which the BCS of one user interacts with the BCSs of other users (e.g., establishing a connection). Our case study includes five optional features, taken from the specifications for the Feature Interaction Contest [5]:

Call Forwarding (CF): The subcriber may specify a forwarding number. All calls to the subscriber will then be forwarded to this number.

Call Forwarding on Busy (CFB): If the subscriber receives a call when she is involved in another call, the feature will redirect the new call to a predetermined forwarding number.

Call Waiting (CW): If the subscriber receives a call when she is involved in another call, she may choose to put the original call on hold, answer the new call, and then toggle between the two calls.

Terminating Call Screening (TCS): This feature allows its subscriber to specify a list of blocked numbers. Any call

originating from a number on this list will be terminated automatically.

Three-Way Calling (TWC): This feature allows a subscriber to add a third user to an existing call. Once three-way communication has been established, any user may chose to leave, resulting in a traditional two-way call configuration.

The BIP2 framework claims to support component-based modelling with an emphasis on inter-component interactions. The primary goal of our case study was to assess these claims in the context of feature-oriented modelling and feature interactions (FIs). We evaluated BIP2's suitability for modelling feature-oriented systems on three main points:

1) **Composed model complexity:** The overall complexity of a complete model of the telephony system (i.e., the BCS together with the user model and optional features for each user).
2) **New feature integration:** The amount of work that a developer must perform to add a new feature to an existing system. We look at the difficulty of design decisions when composing new features in terms of limitations on the number or type of ports in existing components, transitions within the BCS component, and the types of existing connectors. We strive to adhere to the principles of feature-oriented development. That is, the addition of a new feature to the system should not require the modification of the BCS or existing features.
3) **FI Resolution:** The difficulty of detecting and resolving FIs in terms of how the modeller discovers conflicting features and the number of changes they must make in the model to resolve these FIs.

Our secondary goal was to identify design methodologies or patterns for modelling and connecting BIP2 components in feature-oriented systems. In the next section we present three different feature-oriented modelling strategies and evaluate each of them based on the criteria above. For a more complete description of our modelling strategies complete with BIP2 models and code, see our extended technical report [6].

IV. Design Methodologies

Each of our design methodologies approaches the problem of feature composition and integration with the base system in a different way, resulting in different interactions, different degrees of model complexity, and different types of decisions the modeller must make during composition. We give a summary of our evaluations in Table I.

A. Reuse Approach

In the reuse approach, new features are integrated into the base component by reusing existing components and expanding the connectors between basic-call services and users to include the new feature component, and replacing the default interactions with new ones that slightly alter the progression of a call. The inspiration for this approach stems from the idea that a feature overrides existing functionalities provided by the base service. Our case study features can naturally be described in terms of the BCS functionalities they override: CFB, CW, and TWC override the progression of a call when the subscriber is busy, while CF and TCS override the progression of an incoming call.

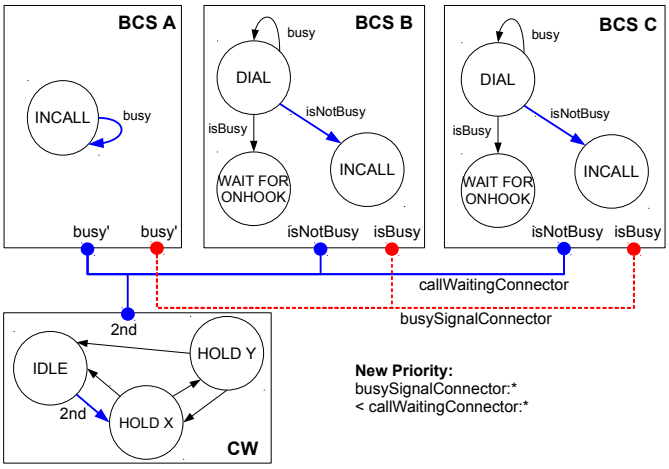

Fig. 1: The integration of CW in the reuse approach (partial models are shown for brevity). The original connector is shown in red and dashed, and the new connector, containing the $2nd$ port from the CW component is shown in blue and solid.

A call progresses through interactions with other basic-call services and users. To integrate a new feature in the reuse approach, we first identify the interactions in the existing components that it overrides. We then expand the connector(s) that contain these interactions to include ports in the new feature's component. Interactions that involve the new feature's synchronizing or triggering ports are then given higher priority than the pre-existing interactions.

We show the integration of CW to an existing BIP2 model in Figure 1. If User A is in a call, the (red) interaction normally terminates subsequent incoming calls by synchronizing the $busy'$ port of User A's BCS with the $isBusy$ port of the caller's BCS, causing the caller to transition to its WAIT FOR ONHOOK place. If User A subscribes to CW, this interaction is replaced with a new interaction (blue) that instead allows the caller to proceed to the INCALL state. The CW feature keeps track of which of the subscriber's calls is currently on hold. The new interaction is given higher priority, thereby replacing the old functionality.

B. Rewire Approach

While the reuse approach allows for the independent development of features and resists changes to the BCS components, the design and integration of a feature is limited by the ports and transitions of existing components. Furthermore, a system with many features that override the same functionalities may result in very large connectors that contain ports from many different components. These connectors are more difficult to specify and define priorities for, as all combinations of enabled ports must be considered. We designed the rewire approach to give the modeller more freedom to modify existing

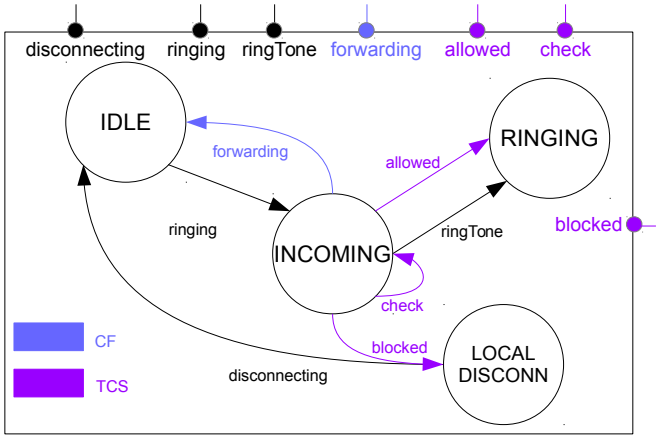

Fig. 2: A BCS component is rewired to support integration with TCS and CF. New transitions and ports for interacting with CF and TCS are shown in blue and purple, respectively.

components with the expectations of easier design decisions and simple components and interaction specifications.

In the rewire approach, new features may entail new functionality (i.e., new ports and transitions) in the pre-existing model of the BCS. When integrating a feature, we first decide the changes the feature makes to the progression of states inside the BCS, and add new transitions and label them with new ports that will be connected to the new feature component. Finally, we design the feature component, and specify the interactions of a new connector that synchronizes transitions in the modified BCS components and the feature component.

In Figure 2, we give an example of the changes made to a BCS component when integrating CF and TCS, both of which modify the progression of an incoming call. In TCS, a new call interacts with the TCS feature component through ports that first *check* and then *allow* or *block* the call. New transitions and new ports (shown in purple) are involved in new interactions with the connected TCS component.

The rewire approach results in feature-specific connectors that are small and similar in behaviour. Fortunately, BIP2 allows modellers to specify *connector types* to ease the specification of many, similar connectors. This further reduces the work of the modeller and the complexity of the overall model in the rewire approach. Unfortunately, the advantages of the rewire approach come at the cost of violating the principles of feature-oriented development: existing components must be extended with new transitions that react to events on new ports.

C. Pipe-and-Filter Approach

The reuse and rewire approaches exemplify the challenge of feature-oriented modelling in BIP. There is a trade-off between modelling freedom versus modularity; by refusing to change existing components, we restrict the ways in which other components can interact with them. To bridge the gap between these two strategies, we adapted an approach that standardizes how components interact with each other.

We took inspiration for our third approach from the Distributed Feature Composition (DFC) architecture developed by Zave and Jackson [7] for the development and composition of telephony features. In DFC, each user's features are connected sequentially in a pipeline, and communications from one user to another propagate through a sequence of features as a call is placed from one BCS to another. Thus, the execution of features is serialized, with each feature triggering the next feature in the pipeline. As a result, DFC provides a default resolution of FIs by imposing a priority ordering on the execution of features, determined by the feature's positions in the sequence (e.g., the last feature in the pipeline provides a final response to a user request).

In our pipe-and-filter approach, we standardize the triggering and synchronizing ports on each feature, making it much easier to interconnect features without knowledge of their internal structure. Synchronized transitions within components are triggered not just by communications on the ports of connectors, but by the specific data conveyed in the communications. Specifically, we designed a new BCS that standardizes the messages that are sent among components. Messages fall into one of three main types: messages that establish a call, busy messages that indicate the other service is currently unavailable, and disconnect messages that indicate one of the participants wishes to terminate a call. Every component has two ports: a synchronizing port in for receiving incoming messages, and a triggering port out for sending outgoing messages. Every interaction between an out and in port passes the following data: (1) The enumerated message type (CONN, BUSY, or TERM), (2) the id of the component that sent the message, and (3) the id of the component that is the designated recipient of the message.

In Figure 3, we show the composition of two BCS components with a TCS feature component. A user's features are arranged and connected in a sequence between her BCS and the feature sequences of other users. Messages "flow" through the pipeline one component at a time. Each component synchronizes with the previous component in the chain; decides whether to react to the received data by modifying the message; and then propagates the message further, either by passing it to the next feature or back to the previous feature.

The standardization of port types and interactions, along features' compliance to the rule that all components must propagate messages either forward or backward through the pipeline, allows features and BCS components to be oblivious of the behaviour and existence of other components, while still reacting predictably to received communications. Features can be designed independently and in parallel. This provides a greater degree of modularity than the rewire approach, which requires modifications to existing components, as well as the reuse approach, which requires knowledge of existing components. Additionally, every feature has the same ports and is linked to other components with the same connectors, further reducing the work of the modeller.

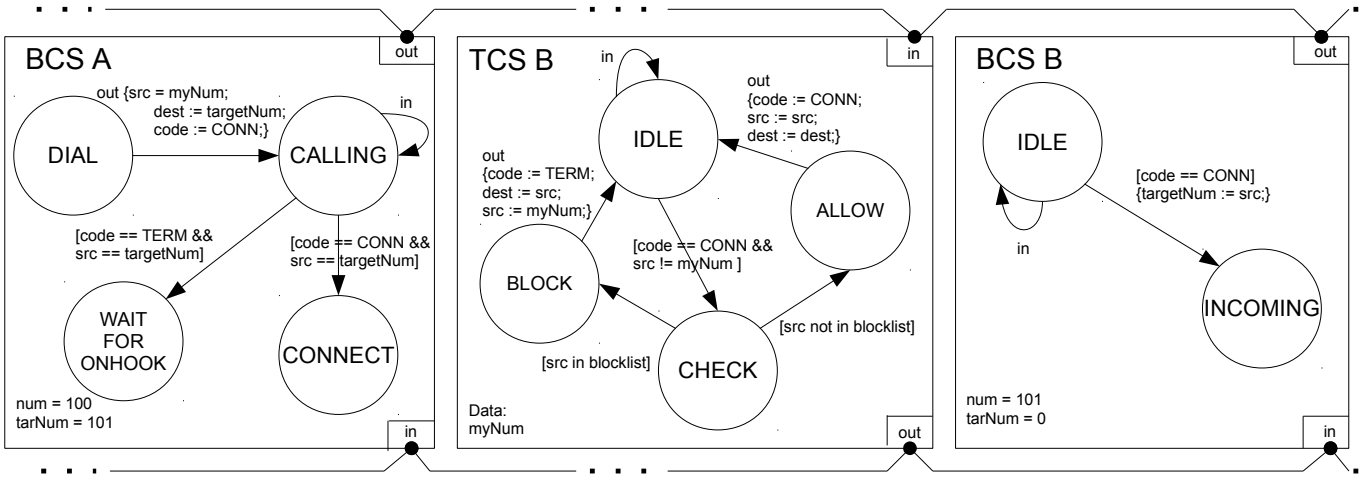

Fig. 3: A partial model of two BCSs and TCS connected in the pipe-and-filter approach. Components are connected sequentially, with interactions that carry connection and tear-down messages as data. Each feature in the pipeline has the ability to modify the messages that pass through it, changing the progression of the call.

D. Discussion

We performed a case study to evaluate each strategy on three main points: the complexity of the overall model (in terms of the number of feature places and transitions, as well as modifications to the BCS and the number and complexity of connectors used to compose the overall model), the work of integrating a new feature into the existing model (in terms of additional feature components, connectors, priorities, and design decisions that require knowledge of existing components), and the difficulty of detecting and resolving feature interactions (in terms of analyzing existing components and the rework required to remove undesired behaviour). We summarized our quantitative data from the case study in Table I.

We found that each approach exhibits complexity in a different aspect of the modelling process, as shown in Table I. The reuse approach has more complex connectors and interaction specifications, whereas the rewire approach adds model complexity in the form of monolithic implementations of features in the BCS component, which violates the principles of feature-oriented design and increases the chance of introducing errors in BCS behaviour. The pipe-and-filter approach introduces complexity in yet another area, requiring more complex feature components to formulate specialized behaviour in response to standardized messages. Feature components in the pipe-and-filter approach require more data variables, and transitions require guard and update functions that react to and modify the component and message data variables.

The integration and resolution of new features require varying amounts of knowledge, work, and design decisions in each of the three approaches. The reuse approach requires the modeller to design new features within the constraints of existing ports and transitions in the BCS. In contrast, the rewire approach affords the modeller more freedom, yet complicates the BCS model and violates the principles of feature-oriented design. In fact, both of our first two approaches require significant knowledge of, and possible modifications to, existing components. In feature-oriented systems with a continuously evolving set of features, it is advantageous for a feature developer to *not* know about the other features in the model. It is this obliviousness and separation of concerns that allows features to be developed in isolation and by third parties, and to be more easily integrated into an existing system without requiring significant rework of existing features or their connectors. The pipe-and-filter strategy is the most effective in supporting feature obliviousness. Not only does every feature have the same interface, but the connector types and their interactions are standardized. What is left to the modeller is to determine the order of connected features in the pipeline, and to instantiate the connectors to realize this pipeline. As a result, the composition of features and resolution of FIs was almost trivial.

We have shown that in BIP2, where specifications of ports, connectors, and interactions require some knowledge of the internal workings and ports of other components, feature-oriented modelling is possible with the pipe-and-filter approach. In this approach, individual features may remain agnostic to other features, only requiring knowledge of the base component during their development and composition.

V. Related Work

Since the framing of the Feature Interaction Problem in 1989 [1], there have been myriad attempts to minimize the effort of the developer in composing systems that are prone to a large number of FIs [2], [8]. Off-line techniques aid the developer during the design and development of the system.

- Techniques for detecting FIs reduce the effort of the developer in discovering problematic compositions of features and pinpointing the sources of undesired behaviour that need to be resolved [9], [10], [11], [12].
- Filtering approaches limit the variability of a system by removing problematic or unlikely combinations of

TABLE I: Comparison of the overall complexity of a fully-composed BIP2 model in each of the three approaches. A fully-composed model has three users, each with a basic-call service, where one user has subscribed to all five optional features.

	Original BCS	Reuse approach	Rewire approach	Pipe-and-filter approach
feature places and transitions	0	29	33	**63**
BCS transitions	39	39	**75**	43
BCS data variables	5	5	5	**8**
defined interactions	6	**66**	6	6
connectors	16	22	**33**	16
priorities	19	26	**48**	0
reworked transitions, interactions, or priorities	0	**8**	5	0

features from analysis, thereby reducing the number of FIs a developer needs to consider to those in a small set of feasible products [13], [14], [15].

On-line techniques for coordinating feature execution resolve FIs as they occur at runtime. Hay and Atlee proposed a specification and composition model that uses feature priority to automatically resolve FIs during composition [16]. Distributed Feature Composition (DFC) developed by Zave and Jackson [7] connects features in a pipe-and-filter architectures, avoiding FIs architecturally by serializing the features' executions. In contrast, BIP provides modellers with the flexibility to specify how features are connected and prioritized.

There have been previous case studies to evaluate the modelling capabilities of BIP. Basu et al. performed a case study on wireless sensor networks [17] to assess the suitability of BIP2 in modelling distributed systems with heterogeneous components. Bourgos et al. conducted a case study on the modelling of a MJPEG decoder [18] to test the use of BIP2 in analyzing the performance of embedded applications on different hardware platforms. While these case studies provide evidence for the flexibility of BIP and its applicability to a wide variety of hardware and software systems, to our knowledge, there are no existing studies that analyze the use of BIP in the context of the Feature Interaction Problem. We provide both an analysis of its use to compose and analyze features and a comparison of design strategies for specifying feature-oriented systems in BIP.

VI. Conclusion

In summary, we investigated the effectiveness of BIP2 for modelling feature-rich systems, with particular attention to the amount of work needed to compose features, the amount of re-work needed to evolve a model to integrate new features, and the degree of complexity of the resulting model. Each of the three strategies that we studied has its advantages and its weaknesses. Ultimately, when considering which strategies help to address the Feature Interaction Problem, the pipe-and-filter approach is the more effective design methodology: (1) it supports and preserves feature modularity, even when new features are added to the system, and (2) the amount of work and re-work needed to add a new feature is substantially less than in the other two strategies.

References

[1] T. Bowen, F. Dworack, C. Chow, N. Griffeth, G. Herman, and Y.-J. Lin, "The feature interaction problem in telecommunications systems," in *Proceedings of the 7th International Conference on Software Engineering for Telecommunication Switching Systems (SETSS)*, 1989, pp. 59–62.

[2] M. Calder, M. Kolberg, E. H. Magill, and S. Reiff-Marganiec, "Feature interaction: a critical review and considered forecast," *Computer Networks: The International Journal of Computer and Telecommunications Networking*, vol. 41, no. 1, pp. 115–141, 2003.

[3] A. Basu, S. Bensalem, M. Bozga, J. Combaz, M. Jaber, T.-H. Nguyen, and J. Sifakis, "Rigorous component-based system design using the bip framework," *IEEE Software*, vol. 28, no. 3, pp. 41–48, 2011.

[4] Verimag, "Bip2 documentation, release 2015.04 (rc7) [online]," Available: http://www-verimag.imag.fr/TOOLS/DCS/bip/doc/latest/pdf/BIP2.pdf (accessed November 5, 2015), Tech. Rep., 2015.

[5] M. Kolberg, E. H. Magill, D. Marples, and S. Reiff-Marganiec, "Second feature interaction contest," in *Proceedings of Feature Interaction Workshop*, M. Calder and E. H. Magill, Eds. IOS Press, 2000, pp. 293–310.

[6] C. Bocovich and J. M. Atlee, "Feature-oriented modelling in BIP: A case study," cs.uwaterloo.ca/~cbocovic/bip.pdf, Tech. Rep. CS-2016-04, 2016.

[7] M. Jackson and P. Zave, "Distributed feature composition: a virtual architecture for telecommunications services," *IEEE Transactions on Software Engineering*, vol. 24, no. 10, pp. 831–847, October 1998.

[8] D. Keck and P. Kuehn, "The feature and service interaction problem in telecommunications systems: a survey," *IEEE Transactions on Software Engineering*, vol. 24, no. 10, pp. 779–796, October 1998.

[9] K. H. Braithwaite and J. M. Atlee, "Towards automated detection of feature interactions," in *Feature Interactions in Telecommunications Systems*. IOS Press, 1994, pp. 36–59.

[10] C. Prehofer, "An object-oriented approach to feature interaction," in *Feature Interactions in Telecommunications Systems IV*. IOS Press, 1997, pp. 313–325.

[11] B. Stepien, "Feature description and feature interaction analysis with use case maps and lotos," in *Feature Interactions in Telecommunications Systems VI*. IOS Press, 2000, pp. 274–289.

[12] S. Apel, A. von Rhein, T. Thüm, and C. Kästner, "Feature-interaction detection based on feature-based specifications," *Computer Networks*, vol. 57, no. 12, pp. 2399 – 2409, 2013.

[13] M. Heisel and J. Souquieres, "A heuristic approach to detect feature interactions in requirements," in *Feature Interactions in Telecommunications Systems V*. IOS Press, 1998, pp. 165–171.

[14] J. Bredereke, "Families of formal requirements in telephone switching," in *Feature Interactions in Telecommunications Systems VI*. IOS Press, 2000, pp. 257–273.

[15] D. O. Keck, "A tool for the identification of interaction-prone call scenarios," in *Feature Interactions in Telecommunications Systems V*. IOS Press, 1998, pp. 276–290.

[16] J. Hay and J. Atlee, "Composing features and resolving interactions," in *Proceedings of ACM SIGSOFT Foundations of Software Engineering (FSE)*, 2000, pp. 110–119.

[17] A. Basu, L. Mounier, M. Poulhiès, J. Pulou, and J. Sifakis, "Using BIP for modeling and verification of networked systems - a case study on Tiny OS-based networks," in *Proceedings of the Sixth IEEE International Symposium on Network Computing and Applications*, Cambridge, USA, July 2007, pp. 257–260.

[18] P. Bourgos, A. Basu, S. Bensalem, K. Huang, and J. Sifakis, "Integrating architectural constraints in application software by source-to-source transformation in BIP," Verimag Research Report, Tech. Rep. TR-2011-1, January 2011.