Runtime Resolution of Feature Interactions through Adaptive Requirement Weakening

Simon Chu
Carnegie Mellon University
Pittsburgh, PA US
cchu2@andrew.cmu.edu

Emma Shedden
University of Michigan
Ann Arbor, MI USA
emshedde@umich.edu

Changjian Zhang, Rômulo Meira-Góes
Carnegie Mellon University
Pittsburgh, PA USA
{changjiz, rmeirago}@andrew.cmu.edu

Gabriel A. Moreno
Software Engineering Institute & Carnegie Mellon University
Pittsburgh, PA USA
gmoreno@sei.cmu.edu

David Garlan, Eunsuk Kang
Carnegie Mellon University
Pittsburgh, PA USA
{dg4d, eunsukk}@andrew.cmu.edu

***Abstract*—The *feature interaction problem* occurs when two or more independently developed components interact with each other in unanticipated ways, resulting in undesirable system behaviors. Feature interaction problems remain a challenge for emerging domains in cyber-physical systems (CPS), such as the Internet of Things and autonomous drones. Existing techniques for resolving feature interactions take a "winner-takes-all" approach, where one out of the conflicting features is selected as the most desirable one, and the rest are disabled. However, when multiple of the conflicting features fulfill important system requirements, being forced to select one of them can result in an undesirable system outcome. In this paper, we propose a new resolution approach that allows all of the conflicting features to continue to partially fulfill their requirements during the resolution process. In particular, our approach leverages the idea of *adaptive requirement weakening*, which involves one or more features temporarily *weakening* their level of performance in order to co-exist with the other features in a consistent manner. Given feature requirements specified in Signal Temporal Logic (STL), we propose an automated method and a runtime architecture for automatically weakening the requirements to resolve a conflict. We demonstrate our approach through case studies on feature interactions in autonomous drones.**

I. Introduction

Modern software systems are often constructed by composing a set of independently developed components or *features*, each of which is designed to achieve a particular objective or a *requirement*. For instance, a typical automotive system contains an array of software features that are designed to ensure vehicle safety under different circumstances, such as emergency braking and lane-keeping assist.

Sometimes, these features can interact with each other in unanticipated ways, resulting in undesirable system behavior. This type of problem, also called the *feature interactions problem*, has been long studied by the software engineering community [1]–[3], but remains a major challenge, especially in emerging domains such as the Internet of Things and autonomous systems [4]–[7]. There are two main aspects to the feature interactions problem: (1) *detection* of a possible conflict between features and (2) its *resolution*. In this paper, we focus on the resolution of feature interactions at *run-time*.

Existing approaches to resolution leverage some notion of what it means for one feature to be more *desirable* than others; then when a conflict arises, the most desirable out of the conflicting features is selected as the one that is ultimately executed by the system (and actions from the rest are discarded). One such common notion is based on user-defined *priorities* or *precedent list* [8]–[11]. Other recent approaches include *variable-specific* resolution (where a conflict between features that modify the same variable is resolved by selecting the action that is considered safest [7], [12]) and *property-based* resolution (where the feature that is most likely to satisfy a given system requirement is selected [13]).

These "winner-takes-all" approaches, however, share one major drawback: When multiple of the conflicting features fulfill important system requirements, being forced to select only one of them leads to an outcome that is undesirable from the developer's perspective. For example, when a pair of conflicting features in a vehicle are both designed to perform critical safety functions (e.g., maneuvering around an obstacle while staying within the lane), discarding one or the other might bring the system into an unsafe state in either case.

To overcome this challenge, we propose a new resolution approach that allows the conflicting features to continue to (partially) fulfill their requirements during the resolution process. The key idea behind this approach is that of *adaptive requirement weakening*: When a feature conflicts with another, it may be acceptable to temporarily "compromise" the level of its functionality, by weakening the requirement that it is designed to achieve. Consider a pair of features, F_1 and F_2, that are designed to fulfill requirements R_1 and R_2, respectively. When in presence of each other, there may be scenarios in which both requirements cannot be fulfilled simultaneously (i.e., $F_1 \oplus F_2 \not\models R_1 \wedge R_2$). To resolve this conflict, instead of disabling F_1 or F_2, our approach involves weakening one or more of the given feature requirements (e.g., from R_1 to R_1') such that the features are able to function and co-exist with each other in a consistent manner (e.g., $F_1 \oplus F_2 \models R_1' \wedge R_2'$). In addition, we say that our approach is *adaptive*, since the degree to which one or more requirements are weakened depends on the particular environmental context in which a conflict arises.

In this paper, we present a realization of this approach in a class of systems called *cyber-physical systems* (CPS), where software features are used to monitor and control one or more physical entities in the environment [14]. The requirements of individual features are specified using a notation called *Signal Temporal Logic (STL)* [15], which is particularly well-suited for describing continuous-domain and time-sensitive behaviors of CPS. Based on the semantics of STL [15], we provide (1) a formal definition of what it means to *weaken* a requirement, (2) a method for automatically transforming a pair of conflicting requirements, R_1 and R_2, into weakened, consistent versions, R_1' and R_2', and (3) a runtime architecture that leverages this method to dynamically modify the behavior of the components and resolves the conflict.

Weakening the requirement of a feature, however, involves degrading the level of its functionality and can reduce the overall utility of the system. Thus, an ideal method would weaken the requirements no more than by a *minimal* degree that is sufficient to resolve the given conflict. Finding such *minimal weakenings*, however, is a challenging problem, since in general, the space of weakening candidates for a given requirement R can be enormous, if not infinite. In particular, we demonstrate how this problem can be formulated as an instance of *mixed-integer linear programming (MILP)* [16], where the goal is to synthesize weakened requirements that no longer conflict with each other while being as close to the original requirements as possible.

We have built a prototype implementation of our runtime resolution approach on top of PX4 Autopilot [17], an open-source autopilot software used in consumer and industrial drones. We have evaluated our approach using four different (possibly conflicting) features in an autonomous drone under a wide range of simulated scenarios. Our evaluation shows that our weakening-based approach is effective at resolving conflicts while allowing the features to continue to satisfy the weakened versions of their requirements.

The contributions of this paper are as follows:

- A theoretical foundation for the resolution of feature interactions using STL-based requirement weakening (Section IV)
- A runtime architecture that leverages requirement weakening to resolve conflicts (Section V),
- An approach for finding minimal weakening through translation into MILP (Section V-B), and;
- An implementation of the weakening-based resolver (Section VI) and an evaluation on case studies involving autonomous drone features (Section VII).

Relevance to SEAMS. Our approach can be regarded as performing a type of self-adaptation, as it involves dynamically changing the behavior of components to manage conflict scenarios that are difficult to predict or resolve at the design time. As opposed to the typical control loop used in self-adaptive systems (e.g., MAPE-K [18]), which adapts the system asynchronously, our approach is synchronous with the control loop of the CPS, as later described in Section V, Fig. 2.

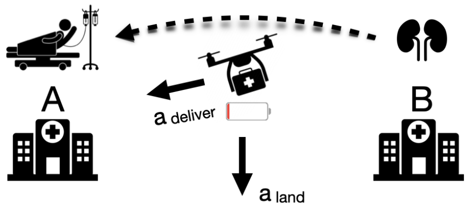

Fig. 1: A possible conflict in an organ delivery drone.

II. Motivating Example

Consider an autonomous organ delivery drone attempting to deliver an organ from a donor in *Hospital B* to a recipient in *Hospital A*, inspired by the example from [19]. The drone contains two features: (1) the *delivery path planner*, which ensures the organ is delivered to the receiver's hospital in the most efficient manner possible, and (2) the *safe landing enforcer*, which ensures that the battery on the drone has enough charge to safely make it to the nearest land. Under normal conditions, the *delivery path planner* will always be active, generating an action ($a_{deliver}$) to move the drone to its destination, while the *safe landing enforcer* remains off by default unless triggered.

Imagine a scenario where the drone encounters unexpected turbulence mid-flight, causing the battery to discharge much faster. When the battery dips below a certain threshold, the safe landing enforcer is activated, which then generates a safe landing action (a_{land}) to direct the drone to the nearest land. Since the delivery path planner is unaware of the landing enforcer, it continues to generate $a_{deliver}$, which results in a conflict between the two features, as depicted in Figure 1.

a) Existing methods: One possible approach to resolving this conflict is to leverage a user-defined list of priorities among the features. The drone operator, for example, may designate the safe landing feature as having the highest priority, and the flight controller could be programmed to disregard the actions from all other features, including the delivery path planner. This approach to resolution results in system behaviors where the requirement of the highest-ranked feature (i.e., safe landing) is satisfied while the others are disregarded; in this example, choosing to land the drone may result in the organ failing to be delivered before its expiry time.

This type of "winner-takes-all" approach, however, may not be suitable in situations where such a strict ordering among features does not exist, or all of the conflicting features play a critical role in maintaining the system's safety and performance. For instance, while landing the drone safely before running out of battery is certainly important, giving up an organ in the process is also a highly undesirable outcome, as the life of a patient may depend on its timely delivery.

b) Proposed method: Our approach, in comparison, attempts to resolve the conflict in a way that satisfies the requirements of both conflicting features. The key idea is that for certain types of requirements in CPS, it may be acceptable to temporarily compromise the degree to which the system satisfies them; i.e., satisfy a weaker version of an original requirement. This notion of requirement satisfaction, also termed *satisficing* [20], can enable a resolution approach that involves relaxing some of the requirements but without entirely giving up on any one of them.

For example, suppose that the requirements for the safe landing and delivery planning features are as follows:

> R_{land}: *If the battery threshold falls below 10%, the drone should land on the nearest land.*
> $R_{deliver}$: *The drone should fly at a fast-enough speed to reach the destination before the delivery time.*

During resolution, one or both of these requirements can be weakened. For example, the requirement for the safe landing feature may be weakened by lowering the threshold that triggers the drone to find a landing spot (e.g., from 10% to 5%). With the new weakened requirement ($R_{land} \rightarrow R'_{land}$), the behavior of the two features become consistent again (i.e., $R'_{land} \wedge R_{delivery}$ is satisfiable). Intuitively, this amounts to delaying the safe landing feature in order to allow the drone to complete its organ delivery mission. As a trade-off, the safe landing feature has compromised the level of safety that it originally promised, since there is now an increased risk that the drone may run out of battery before landing.

Alternatively, the conflict could be resolved by weakening both requirements. For example, $R_{delivery}$ may be weakened by reducing the speed of the drone, to allow the battery to be depleted at a slower rate than at the original speed. This would cause a delay in organ delivery, but it would also enable the drone to complete its delivery while weakening the battery threshold from 10% to 8% only. In either case, this weakening-based approach results in an arguably more desirable system outcome than the priority-based method, since both the requirements of the features can be satisfied (at the cost of temporarily compromising their optimal functionality).

c) Challenges: In general, there may be a large number of ways to resolve a conflict using this approach, weakening one or more requirements by different amounts. Since weakening involves degrading the functionality of the features, an ideal resolution process would involve weakening the requirements no more than needed. At the same time, the system operator may also wish to place a harder constraint on the maximum amount by which a requirement can be weakened (e.g., for R_{land}, the threshold cannot be set below 5%, since that might compromise the safety of the battery beyond what is acceptable). We later show (1) how this type of weakening can be formally realized over STL and (2) an approach that uses a MILP solver to generate minimal weakenings.

III. Preliminaries

a) Signals: In our approach, the behavior of CPS is modeled by real-valued continuous-time *signals*. Formally, a signal s is a function $\mathbf{s} : T \rightarrow D$ mapping from a time domain, $T \subseteq \mathbb{R}_{\geq 0}$, to a tuple of k real numbers, $D \subseteq \mathbb{R}^k$. Intuitively, the value of a signal $\mathbf{s}(t) = (v_1, \ldots, v_k)$ represents different state variables of the system at time t; e.g., v_1 might represent the altitude of the drone.

b) Signal Temporal Logic (STL): STL extends linear temporal logic (LTL) [21] for specifying the time-varying behavior of a system in terms of signals. The basic unit of formula in STL is a signal predicate in the form of $f(\mathbf{s}(t)) > 0$, where f is a function from D to $\mathbb{R}$; i.e., the predicate is true if and only if $f(\mathbf{s}(t))$ is greater than zero. Then, the syntax of an STL formula φ is defined as:

$$\varphi := f(\mathbf{s}(t)) > 0 \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \mathcal{U}_{[a,b]} \varphi_2$$

where $a, b \in \mathbb{R}$ and $a < b$. The *until* operator $\varphi_1 \mathcal{U}_{[a,b]} \varphi_2$ means that φ_1 must hold until φ_2 becomes true within a time interval $[a, b]$. The until operator can be used to define two other important temporal operators: *eventually* ($\Diamond_{[a,b]}\varphi := True\ \mathcal{U}_{[a,b]}\varphi$) and *always* ($\Box_{[a,b]}\varphi := \neg\Diamond_{[a,b]}\neg\varphi$).

c) Robustness: Typically, the semantics of temporal logic such as LTL is based on a *binary* notion of formula satisfaction (i.e., formula φ is either satisfied or violated by the system). Due to its signal-based nature, STL also supports a *quantitative* notion of satisfaction, which allows reasoning about how "close" or "far" the system is from satisfying or violating a property. This quantitative measure is also called the *robustness* of satisfaction.

Informally, the robustness of signal $\mathbf{s}$ with respect to formula φ at time t, denoted by $\rho(\varphi, \mathbf{s}, t)$, represents the smallest difference between the actual signal value and the threshold at which the system violates φ. For example, if the property φ says that "the drone should maintain an altitude of at least 5.0 meters," then $\rho(\varphi, \mathbf{s}, t)$ represents how close to 5.0 meters the drone maintains its altitude. Formally, robustness is defined over STL formulas as follows:

$$\begin{aligned}
\rho(f(\mathbf{s}(t)) > 0, \mathbf{s}, t) &\equiv f(\mathbf{s}(t)) \\
\rho(\neg\varphi, \mathbf{s}, t) &\equiv -\rho(\varphi, \mathbf{s}, t) \\
\rho(\varphi_1 \wedge \varphi_2, s, t) &\equiv \min\{\rho(\varphi_1, \mathbf{s}, t), \rho(\varphi_2, \mathbf{s}, t)\} \\
\rho(\Diamond_{[a,b]}\varphi, \mathbf{s}, t) &\equiv \sup_{t_1 \in [t+a, t+b]} \rho(\varphi, \mathbf{s}, t_1) \\
\rho(\Box_{[a,b]}\varphi, \mathbf{s}, t) &\equiv \inf_{t_1 \in [t+a, t+b]} \rho(\varphi, \mathbf{s}, t_1)
\end{aligned}$$

where $\inf_{x \in X} f(x)$ is the greatest lower bound of some function $f : X \rightarrow \mathbb{R}$ (and sup the least upper bound). The robustness of satisfying predicate $f(\mathbf{s}(t)) > 0$ captures how close signal $\mathbf{s}$ at time t is above or below zero. For example, consider formula $\varphi \equiv alt(t) - 5 > 0$, capturing the property that "the drone altitude is at least 5.0 meters." If, at time t, the altitude signal is $alt(t) = 10$ meters (i.e., 5.0 meters above the required altitude), $\rho(\varphi, \mathbf{s}, t)$ is computed as 5.

On the other hand, robustness $\rho(\Box\varphi, a, t)$ describes the point at which the system is furthest away from satisfying φ. For instance, consider property $\phi \equiv \Box_{[0,2]}(alt(t) - 5 > 0)$, which says that the drone must maintain a minimum altitude of 5.0 meters for interval $t = [0, 2]$. Suppose that the system evolves to generate signal $\mathbf{s}$ with the altitude of 6.0, 3.0, 5.5 meters at $t = 0, 1, 2$, respectively; then, $\rho(\phi, \mathbf{s}, 0) =$

$\rho(alt(t) - 5 > 0)$,$\mathbf{s}, 1)$ = -2.0 (i.e., the system *violates* ϕ by the robustness value of 2.0).

IV. Requirement Weakening

We present an extension to STL to support the systematic weakening of requirements in STL. It enables us to specify the maximum extent of weakening allowed for a requirement and quantitatively measure the degree of weakening, which later plays an important role to ensure that requirements are weakened no more than needed to resolve a conflict. The extension is built upon the observation that the robustness value of an instantiated STL formula can be increased by changing the atomic proposition, and shortening or enlarging the time interval of the temporal operators in STL. This increase in robustness quantification leads to a less restrictive requirement, which is easier to satisfy than the original one.

A. weakSTL: STL with Weakening

We propose *weakSTL*, an extension to STL with weakening semantics. *weakSTL* introduces additional parameters to atomic propositions and temporal operators to indicate the range of signal values and time intervals allowed for weakening. The syntax of a *weakSTL* formula is defined as:

$$\begin{aligned}\varphi \;:=\; & true \mid f_p(\mathbf{s}(t)) > 0 \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \\ & \Box_{I,p,q}\varphi \mid \Diamond_{I,p,q}\varphi \mid \varphi\, \mathcal{U}_I\, \psi\end{aligned}$$

where I represents the original time interval $[a, b]$, and p, q represent the weakening parameters that define the maximum range for the signal values and time intervals to be weakened. Note that, our definition does not allow weakening parameters over the *Until* operator ($\varphi\, \mathcal{U}_{[a,b]}\, \psi$) because when φ and ψ are non-trivial formulas, changing the time interval ($[a, b]$) has a mixed-effect over the difficulty of satisfaction; i.e., it cannot guarantee the resulting formula will be weaker. For example, increasing b allows ψ to occur later (which makes it easier to satisfy) but requires φ to hold longer (which makes it harder).

1) Semantics: Formally, a *weakSTL* formula φ defines a set of STL formulas that are weaker variants of the original STL formula. We define the translation rules $M : weakSTL \to \mathbb{P}(STL)$ as follows:

$$\begin{aligned}M(true) &= true\\ M(f_p(\mathbf{s}(t)) > 0) &= \{f(\mathbf{s}(t)) + i > 0 \mid 0 \le i \le p\}\\ M(\neg\varphi) &= \{\neg\varphi' \mid \varphi' \in M_s(\varphi)\}\\ M(\varphi \wedge \psi) &= \{\varphi' \wedge \psi' \mid \varphi' \in M(\varphi), \psi' \in M(\psi)\}\\ M(\varphi \vee \psi) &= \{\varphi' \vee \psi' \mid \varphi' \in M(\varphi), \psi' \in M(\psi)\}\\ M(\Box_{[a,b],p,q}\ \varphi) &= \{\Box_{[a+i,b-j]}\ \varphi' \mid \\ &\qquad 0 \le i \le p, 0 \le j \le q, \varphi' \in M(\varphi)\}\\ M(\Diamond_{[a,b],p,q}\ \varphi) &= \{\Diamond_{[a-i,b+j]}\ \varphi' \mid \\ &\qquad 0 \le i \le p, 0 \le j \le q, \varphi' \in M(\varphi)\}\\ M(\varphi\, \mathcal{U}_{[a,b]}\ \psi) &= \{\varphi'\, \mathcal{U}_{[a,b]}\ \psi' \mid \varphi' \in M(\varphi), \psi' \in M(\psi)\}\end{aligned}$$

For example, consider $\varphi \equiv f_p(\mathbf{s}(t)) > 0$ for some given value p, where $f(\mathbf{s}(t)) \equiv alt(t) - 5$. Then, $M(\varphi)$ represents the set of all STL formulas of the form $alt(t) - 5 + i > 0$ for $i \le p$; in other words, $M(\varphi)$ represents weaker variants of φ that requires the drone to maintain an altitude higher than $(5 - i)$ only instead of 5 meters in the original φ.

Given $\varphi \equiv \Box\phi$, $M(\varphi)$ represents the set of formulas where the system is required to maintain ϕ throughout a *narrower* interval than the one specified by the original formula φ. Similarly, for $\varphi \equiv \Diamond\phi$, $M(\varphi)$ relaxes this requirement to allow the system to satisfy ϕ just once during a larger window than in φ, which is easier to fulfill.

Note that weakening formulas of the form $\neg\varphi$ involves *strengthening* φ. To achieve this, we also introduce M_s, which defines the translation rules for strengthening a STL formula. In short, M_s is similar to M but inverses the weakening computations in M:

$$\begin{aligned}M_s(f_p(\mathbf{s}(t)) > 0) &= \{f(\mathbf{s}(t)) - i > 0 \mid 0 \le i \le p\}\\ M_s(\neg\varphi) &= \{\neg\varphi' \mid \varphi' \in M(\varphi)\}\\ M_s(\Box_{[a,b],p,q}\ \varphi) &= \{\Box_{[a-i,b+j]}\ \varphi' \mid \\ &\qquad 0 \le i \le p, 0 \le j \le q, \varphi' \in M_s(\varphi)\}\\ M_s(\Diamond_{[a,b],p,q}\ \varphi) &= \{\Diamond_{[a+i,b-j]}\ \varphi' \mid \\ &\qquad 0 \le i \le p, 0 \le j \le q, \varphi' \in M_s(\varphi)\}\end{aligned}$$

2) Instantiation: By assigning concrete weakening values to each of the weakened operators and propositions, we can instantiate an STL formula from a *weakSTL* formula. Formally, we state this as $\varphi_\theta = \kappa(\varphi, \theta)$, where κ is the instantiation function, φ is a *weakSTL* formula, φ_θ is an STL formula, and θ assigns concrete weakening values within the range given by the weakening parameters (i.e., p, q's) in φ.

Specifically, θ is as a partial function of type $weakSTL \to \mathbb{Z}^k$, where k equals the total number of weakening parameters in φ. Given φ, θ maps each sub-expression of φ to concrete values that are used to determine how much that sub-expression is weakened. Formally, we have:

$$\begin{aligned}\theta(true) &= ()\\ \theta(f_p(\mathbf{s}(t)) > 0) &= (x),\ 0 \le x \le p\\ \theta(\neg\varphi) &= \theta(\varphi)\\ \theta(\varphi \wedge \psi) &= \theta(\varphi)^\frown\theta(\psi)\\ \theta(\varphi \vee \psi) &= \theta(\varphi)^\frown\theta(\psi)\\ \theta(\Box_{[a,b],p,q}\ \varphi) &= (x, y)^\frown\theta(\varphi),\ 0 \le x \le p \wedge 0 \le y \le q\\ \theta(\Diamond_{[a,b],p,q}\ \varphi) &= (x, y)^\frown\theta(\varphi),\ 0 \le x \le p \wedge 0 \le y \le q\\ \theta(\varphi\, \mathcal{U}_{[a,b]}\ \psi) &= \theta(\varphi)^\frown\theta(\psi)\\ &. \end{aligned} \tag{1}$$

Then, the instantiation function κ takes a given mapping θ and generates an STL formula based on the translations rules M and M_s, i.e., plugging in values from θ into i, j's as it walks over the sub-expressions of φ recursively.

We say that a *weakSTL* formula φ is satisfiable when there exists an instantiated STL formula that is satisfiable, i.e.,

$$(\mathbf{s}, t) \models \varphi \Leftrightarrow \exists\varphi_\theta \in M(\varphi)\ \bullet\ (\mathbf{s}, t) \models \varphi_\theta$$

In addition, we define φ_0 be the instantiation without any weakening from the *weakSTL* formula φ, i.e., θ is defined to be 0 for every sub-expression of φ.

Finally, we leverage the robustness concept in the original STL to measure the *degree of weakening* between two instantiations of a *weakSTL* formula, as follows:

$$\Delta(\varphi_{\theta 1}, \varphi_{\theta 2}, \mathbf{s}, t) = \rho(\varphi_{\theta 2}, \mathbf{s}, t) - \rho(\varphi_{\theta 1}, \mathbf{s}, t)$$

where $\varphi_{\theta 1}, \varphi_{\theta 2} \in M(\varphi)$.

3) Example: Let us revisit the example in Section III, where $\phi \equiv \Box_{[0,2]}(alt(t) - 5 > 0)$. Consider the *weakSTL* formula $\varphi \equiv \Box_{[0,2],0,2}(alt(t) - 5 > 0)$ (i.e., $\varphi = M(\phi)$ with $p = 0$ and $q = 2$). Then, φ can be instantiated into two weaker versions of φ, by weakening the time interval to $t' = [0, 1]$ with $\theta = (0, 1)$ or $t' = [0, 0]$ with $\theta = (0, 2)$.

Given signal $\mathbf{s}$ with an altitude of 6.0, 3.0, 5.5 meters at $t = 0, 1, 2$, respectively, the original STL requirement ϕ is violated, with the robustness value of -2.0. However, with $\theta = (0, 2)$, the resulting, weaker STL requirement φ_θ is satisfied, with the robustness value of $\rho(\varphi_\theta, \mathbf{s}, 0) = 1.0$. The degree of weakening between the weaker and initial requirements is measured as $\Delta(\varphi_0, \varphi_\theta) = \Delta(\phi, \varphi_\theta) = 1.0 - (-2.0) = 3.0$.

V. Runtime Resolution Architecture

An overview of our proposed runtime architecture for weakening-based resolution is shown in Figure 2. We assume that each feature in our system periodically observes the state of the environment (through one or more sensors) and generate a command to an actuator in order to influence the state. For example, the safe landing feature from our running example periodically monitors the state of the battery (which is a physical component and is thus considered part of the environment for the software system. If the battery level drops below a safe threshold, the landing feature generates a command to direct the drone to land on the nearest land.

There are two major components in the proposed architecture: (1) the *detector*, which detects possible conflicts between the features, and (2) the *resolver*, which resolves a possible conflict by generating a new set of actions that are consistent with each other. Since the focus of this paper is on resolution, for detection, we adopt the approach proposed by [7] and [12], where a pair of features are considered to be in conflict if they generate actions that modify an overlapping part of the environment (e.g., the safe landing and delivery path planning features both affect the direction of the drone's next movement and thus are in conflict).

When triggered by the detector, the resolver takes three types of inputs: (1) a conflict, represented by a set of conflicting features, (2) a set of requirements for the conflicting features, and (3) an *environment model*, which describes how the state of the environment evolves based on the action of the system (explained further below). Then, the resolver performs two steps: (1) *weakening* of one or more of the given requirements, and (2) produce a set of actions of the features that adhere to the weakened requirements (and thus, non-conflicting with each other).

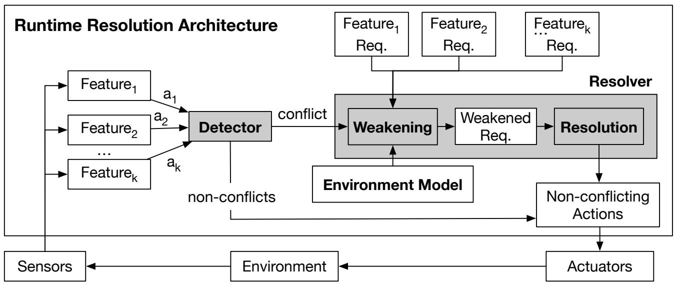

Fig. 2: Overview of the proposed resolution architecture.

In the subsequent sections, we further explain (1) how an environment model is used to predict the behavior of a system given an action and (2) how the weakening and resolution steps by the resolver are carried out using a MILP solver.

A. Environment Model

Given a set of feature requirements, $R_1, ..., R_k$, the goal of the resolution is to find weaker versions of one or more of them, such that they both are consistent (i.e., it is possible to satisfy $R'_1 \wedge ... \wedge R'_k$). In the context of STL, checking the satisfaction of weakened requirement R' involves evaluating it over some signal $\mathbf{s}$ that represents the possible future states of the system *if* the component behaved according to R'. In the proposed framework, the environment model plays the role of generating such a *predictive signal*.

More specifically, the environment model is represented as transition system $T = (\mathcal{Q}, \mathcal{A}, \delta, \mathcal{Q}_i)$, where:

- $\mathcal{Q} \subseteq \mathbb{R}^k$ is the set of environment states. Each state is a particular combination of values for signal variables, represented as a k-dimensional tuple; $q = (v_1, ...v_k) \in \mathcal{Q}$.
- $\mathcal{A}$ is the set of actuator actions.
- $\delta : \mathcal{Q} \times \mathcal{A} \to \mathcal{Q}$ is the transition function that captures how the system moves from one state to another by performing an action.
- $\mathcal{Q}_i$ is the set of initial states.

For example, the environment model for the drone example may capture the (x, y, z) location of the drone, its velocity, as well as the amount of remaining battery. The location of the drone changes during each transition depending on the current velocity, which, in turn, may be modified by a system action that accelerates or decelerates the drone. Similarly, the battery level also can be modeled as decreasing at a steady rate ($drain\_rate$) while the drone is in movement:

$$q' = \delta(q, a)$$
$$q'.battery\_level = q.battery\_level - drain\_rate$$

Then, given a sequence of actions $a_1, a_2, ..., a_n$ and the current state q, the environment model can be executed over these actions to generate a corresponding state sequence, $q_0, q_1, ...q_n$, which can then be formed into predictive signal $\mathbf{s}$.

Our approach does not prescribe a particular notation for specifying an environment model, as long as it can be used to generate signals as depicted above. For our implementation, we use the MiniZinc modeling language [22], which provides declarative constraints for specifying relationships between different variables of a system.

B. Weakening-based Resolution as MILP

In our approach, the weakening and resolution steps inside the resolver (Figure 2) are carried out together by reduction to a constrained optimization problem—in particular, MILP. A standard MILP problem involves finding values for a set of decision variables that maximize (or minimize) an objective function while satisfying a set of constraints. We describe how our problem can be formulated into MILP[1].

1) Minimal requirements: As inputs, the resolver is given a pair of feature requirements in STL, R_1 and R_2. From these, the resolver first generates $weakSTL$ formulas for R_1 and R_2; i.e., $\varphi = M(R_1)$ and $\psi = M(R_2)$. In addition, the user specifies the maximum allowed degree of weakening by providing values for p and q for each *weakSTL* requirement. These bounds, in effect, define the weakest possible requirement that is allowed by the *weakSTL* requirement, also called the *minimal requirement*.

For example, consider $R_1 \equiv (alt(t) > 5)$. Suppose that the user is willing to accept R_1 to be weakened but no more than $(alt(t) > 2)$, since staying below that altitude is considered unacceptable. Then, the user would specify the value of 3 for bound p in *weakSTL* $\varphi \equiv M(R_1) = f_p(\mathbf{s}(t)) > 0$.

2) Optimization problem: Weakening-based resolution is formulated as a constrained optimization problem, as follows:

Problem 1: Given $\varphi, \psi \in weakSTL$, transition system $T = (\mathcal{Q}, \mathcal{A}, \delta, \mathcal{Q}_i)$, and state sequence $q^t = q_0, \ldots, q_t$ representing the signal observed from the environment so far, compute:

$$\text{argmin}_{\theta(\varphi),\theta(\psi),\mathbf{a}} \quad \Delta(\varphi_0, \varphi_{\theta(\varphi)}, \mathbf{s}, 0) \; + \; \Delta(\psi_0, \psi_{\theta(\psi)}, \mathbf{s}, 0) \tag{2}$$

$$s.t. \quad \theta(\varphi) \in \mathbb{Z}^k \tag{3}$$

$$\theta(\psi) \in \mathbb{Z}^m \tag{4}$$

$$\mathbf{s}_i = q_i \text{ for } i \leq t \tag{5}$$

$$\mathbf{s}_i = \delta(s_{i-1}, \mathbf{a}_{i-1}) \text{ for } t < i \leq t+N \tag{6}$$

$$(\mathbf{s}, 0) \models \varphi_{\theta(\varphi)} \wedge \psi_{\theta(\psi)} \tag{7}$$

where k and m are the total number of weakening parameters of φ and ψ as defined by Eq. 1, $N \in \mathbb{N}$ is a finite horizon provide by the user, $\mathbf{s} = s_0 \ldots s_{t+N}$, and $\mathbf{a} = a_t \ldots a_{t+N-1}$. Moreover, transition system T is provided by the environment model and is used to predict signal $\mathbf{s}$ from t to $t+N$.

Problem 1 poses the weakening resolution problem as an optimization problem. Intuitively, this optimization problem generates: (1) two weakening parameters $\theta(\varphi), \theta(\psi)$ for weakSTL formulas φ, ψ; and (2) a sequence of N actions $a_t \ldots a_{t+N-1}$ that result in the satisfaction of the weakened requirements. These parameters are generated such that the degree of weakening for each of the weakSTL formulas is minimized, i.e., weaken the requirements no more than necessary (Eq 2). Moreover, these parameters must satisfy conditions over the predicted signal $\mathbf{s}$ as well as on the satisfiability of the weakened STL requirements, $\varphi_{\theta(\varphi)} \wedge \psi_{\theta(\psi)}$.

Signal $\mathbf{s}$ captures both the past state values as well as the predicted states using the environment model T. To ensure that signal $\mathbf{s}$ and the satisfaction of the weakened STL formulas, we add five constraints to the optimization problem, Eqs. 3-7. Equations 3-4 restrict the weakening parameters based on the *weakSTL* φ and ψ. Equation 5 guarantees that the past state values cannot be changed, i.e., we cannot change the past behavior. Next, Eq. 6 ensures that the predicted state values are generated via the transition system T. Lastly, we ensure that the weakened STL formulas are satisfied, i.e., the conflict has been resolved (Eq. 7).

To encode Problem 1 as an MILP instance, we extend the MILP-based formulation of the STL control synthesis problem in [23] to capture the degree of weakening $\Delta(\varphi_0, \varphi_{\theta(\varphi)}, \mathbf{s}, 0)$ and $\Delta(\psi_0, \psi_{\theta(\psi)}, \mathbf{s}, 0)$. In [23], the control synthesis problem finds a signal $\mathbf{s}$ that satisfies a given STL formula under an environmental model. In our scenario, we include additional variables to capture the weakening parameters $\theta(\varphi)$ and $\theta(\psi)$ to the STL MILP encoding in [23]. As in [23], we assume that every predicate function f that defines $f(\mathbf{s}(t)) > 0$ in STL formula φ must be a linear or affine function. This assumption guarantees that our encoding is expressible as a MILP.

Finally, the translated MILP problem is dispatched to an off-the-shelf solver (Gurobi [24] in our implementation). If the solver is able to find a solution, the resolver returns the generated action sequence $\mathbf{a} = a_t \ldots a_{t+N-1}$ to be executed by the system as the resolved actions.

VI. Implementation

A. Simulator

To demonstrate our approach, we have implemented a prototype of our resolution architecture[2] on top of PX4, an open source flight control software [25]. To run our experiments, we use the jMAVSim drone simulator (part of PX4), which supports the simulation of the physical dynamics of the drone while it reacts to control actions.

For evaluation, we implemented the following features on top of the PX4 flight control software:

- *Delivery Planning Feature*: Plans for the shortest path from point A to point B and generates a set of velocity vectors during the flight to follow the designated path.
- *Safe Landing Feature*: Directs the drone to land on the nearest land when the battery level drops below a preset safety threshold.
- *Boundary Enforcer*: Ensures that the drone remains within the map boundaries. When active, it generates a velocity vector orthogonal to the map boundary.
- *Runaway Enforcer*: Ensures that the drone stays away from drones nearby. When active, it generates a velocity vector to evade a nearby drone.

[1]Without loss of generality, we show a formulation for a MILP problem to resolve a conflict between *two* features.

[2]All of the code, models, and experimental data is available at https://github.com/sychoo/CPS-weakening-based-resolution

B. Environment Model

As discussed in Section V, our framework leverages a model of the environment during the resolution process. For the drone system, the environment model captures (1) 3D Cartesian space around the drone, (2) the physical dynamics of the drone, including how its location is changed by an action that sets its velocity, and how its speed is affected by an acceleration/deceleration action, (3) the amount of remaining battery and its depletion rate (based on the velocity of the ego drone), and (4) the estimated speed and location of a nearby drone (which we call the *chaser* drone); in particular, the model assumes that the chaser is moving towards the ego drone at a fixed speed. We believe that this model is general enough to capture important aspects of a typical drone environment and reusable across multiple features.

The environment model is specified in the MiniZinc modeling language and is automatically translated into the underlying MILP constraints during the resolution process.

VII. Evaluation

This section presents the evaluation of our proposed approach. We focus on the following 3 research questions:

- **RQ1**. Does the weakening-based approach better achieve the desired feature requirements compared to an existing approach?
- **RQ2**. Does the weakening-based approach provide a stronger guarantee for satisfying the minimal feature requirements compared to an existing approach?
- **RQ3**. What is the performance overhead of our approach? Does it interfere significantly with the system operation?

To study the proposed questions, we conducted two case studies involving autonomous drones: (1) an organ delivery drone and (2) a surveillance drone. In each case study, we evaluated our weakening-based method to resolve conflicts between two different features of the drone.

A. Experimental Design

We designed a set of experiments to compare the proposed weakening-based approach to a priority-based resolution approach, which uses a fixed ordering among the features and only selects the action generated by the highest-ranking feature. The weakening-based resolution approach does not require any ordering between different features. Instead, it allows the user to define an original requirement and a minimal requirement for each feature (specified as bounds on *weakSTL* formulas, as described in Section V-B).

When the MILP solver is unable to generate a solution, it outputs UNSAT (for unsatisfiability); this may occur in certain environmental contexts where it is impossible to weaken the requirements (within the maximum allowed degrees as defined by the minimal requirement). In that case, the resolver simply selects one of the conflicting actions to execute.

To test the resolution approaches under diverse scenarios, we randomly generated different configurations for the drone simulation, including the initial starting points of the ego and nearby drones, the maximum speeds of the drones, the mission waypoints (e.g., delivery destination), and the size of the map boundary. Then, for each of these scenarios, we simulated the drone multiple times (each under a different resolution approach) and recorded the system states throughout its execution (i.e., the signal for the entire simulation).

To measure the performance of the resolution approaches, we use the robustness of satisfaction of the given feature requirements as the metric. Our rationale behind choosing this metric is that robustness captures how well the system is achieving the objectives of the features, and thus can serve as a reasonable proxy for the desirability of system behavior that results from a particular resolution approach. In particular, to analyze the impact of resolution, we measure and record the robustness values for the *original* (not the weakened) requirements during the period of a feature interaction; that is, we start collecting the values starting at the time point where both features are activated and stop at the point where both features are deactivated.

Finally, before carrying out our experiments, we developed the following hypotheses to be tested:

- **H1 (for RQ1).** The weakening-based approach results in a higher overall satisfaction of feature requirements than the priority-based approach.
- **H2 (for RQ2).** The weakening-based approach provides a stronger guarantee of minimal requirements than the priority-based approach.
- **H3 (for RQ3).** The weakening-based approach incurs non-trivial overhead, but it is not significant enough to disrupt the operation of the existing drone controller.

All our experiments were run on a macOS machine with 32 GB RAM and a 6-core Intel Core i7.

B. Organ Delivery Drone Case Study

As described in Section II, there are two features of interest in the organ delivery drone: the *delivery planner* and the *the safe landing feature*. A conflict between these two features arises when both features are activated simultaneously (i.e., the battery drops below a safe threshold while the planner is computing the next velocity vector to the destination).

The original STL requirements specified for the two features are as follows:

$$R_{deliver} : \Box_{[0,1]}(curr\_speed > \\ (distance\_to\_dest/remaining\_delivery\_time))$$
$$R_{land} : \Box_{[0,1]}(battery < 40\% \rightarrow \Diamond_{[0,1]}(is\_landing = 1))$$

In addition, we also specified the following minimal requirements for the two features:

$$R_{deliver} : \Box_{[0,1]}(curr\_speed > \\ (distance\_to\_dest/remaining\_delivery\_time))$$
$$R_{land} : \Box_{[0,1]}(battery < 20\% \rightarrow \Diamond_{[0,1]}(is\_landing = 1))$$

Note that the minimal requirement for the delivery planner is the same as the original one (i.e., it cannot be weakened), since timely delivery of the organ is considered critical.

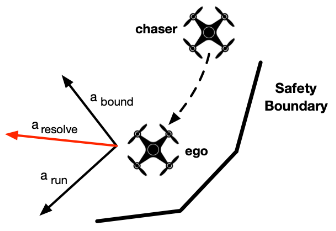

Fig. 3: A conflict scenario between the boundary and runaway enforcers. Action $a_{resolve}$ represents a possible action generated by the weakening-based approach.

For this case study, we generated 25 randomized scenarios by varying the configuration parameters as described in Section VII-A. Then, we ran each scenario four times: (1) weakening-based approach with the delivery planner as the fallback action, (2) weakening with the landing feature as the fallback, (3) priority-based approach with the planner as the preferred feature, and (4) priority with the landing feature preferred. This resulted in a total of 100 scenario runs.

C. Surveillance Drone Case Study

Consider a drone that performs a surveillance mission, visiting a set of waypoints within a designated boundary of the map (inspired by an example from [26]). The environment contains another simulated drone (called the *chaser*) that constantly flies towards the ego drone. The two features of interest here are the *boundary enforcer* and the *runaway enforcer*, each of which is tasked with keeping the drone safe from a collision with the boundary or the chaser (respectively). A conflict can occur in situations when the ego drone travels to a position that is close to the boundary and the chaser, as shown in Figure 3.

The following original STL requirements were specified for the two features:

$$R_{runaway} : \square_{[0,1]}(distance\_to\_chaser > 10)$$
$$R_{boundary} : \square_{[0,1]}(distance\_to\_boundary <= 20 \rightarrow \Diamond_{[0,1]} distance\_to\_boundary > 20)$$

In addition, we also specified the following minimal requirements for the two features:

$$R_{runaway} : \square_{[0,1]}(distance\_to\_chaser > 2)$$
$$R_{boundary} : \square_{[0,1]}(distance\_to\_boundary <= 20 \rightarrow \Diamond_{[0,1]} distance\_to\_boundary > 2)$$

As with the organ delivery case study, we generated 25 random scenarios and ran each scenario four times (twice with the weakening-based approach and the other two times with the priority-based approach).

D. Experimental Result

Figure 4 shows, for each case study, the *overall* robustness values for the weakening-based and priority-based approaches. The overall robustness value is computed as the average of the normalized robustness values for the feature requirements, and is intended to show how the system fulfills the objectives of all the features. As seen in Figure 4, the weakening-based approach achieves higher overall robustness than the priority-based approach, as it attempts to satisfice the requirements of both features (unlike the priority-based method, which gives up on the feature that is not selected).

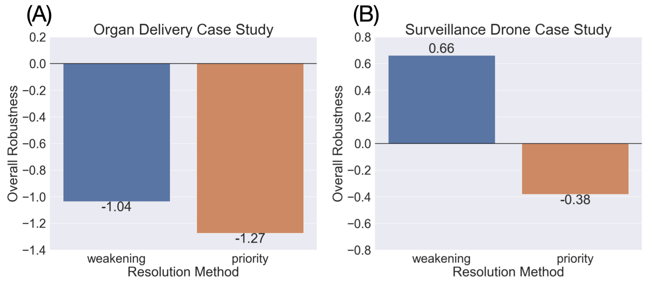

Fig. 4: Overall robustness values for the two case studies.

One may note that in the organ delivery case study, the overall robustness values for both approaches are negative; this is because during the conflicts in these scenarios, the actions available to the drone are drastically different (land vs. keep flying) and thus selecting one feature will result in a large violation of the other's requirement. Even in such negative scenarios, the weakening-based approach is still able to achieve a lower overall violation of the requirements than the priority-based method, as shown in Figure 4.

We provide a more detailed analysis of the results, broken down by the individual feature requirements, shown in Fig. 5.

1) Organ delivery drone: In Fig. 5, charts (A) and (B) show the average robustness values for the landing and delivery requirements, respectively. The results show that the priority-based approach achieves the highest robustness for the requirement of the feature that it selects (e.g., 7.07 for the delivery planning feature in (B)). At the same time, the requirement of the feature that is discarded by the priority-based method shows a large violation. Both of these outcomes are as expected, since the system achieves the requirement of the feature that it specifically prioritizes.

In comparison, the results suggest that the weakening-based approach achieves a compromise between prioritizing one of the features and discarding the other. For example, in chart (B), although the weakening-based approach achieves a lower robustness than the priority method that selects the delivery feature, it avoids the large violation that would result if this feature was entirely discarded.

In (C), it can be seen that the weakening-based approach ensures the satisfaction of the minimal requirement (even in the worst case) while the priority-based method fails to do so. Lastly, in (D), none of the resolution methods can guarantee the delivery planning feature, as it is deemed as a hard constraint that cannot be weakened (i.e., the minimal requirement is the same as the original requirement). We did, however, observe that the weakening-based resolution still obtains a higher robustness in its worst-case scenario than the priority-based method does.

2) Surveillance drone: Similar patterns can be observed here as the ones in the prior case study. In Fig. 5, charts

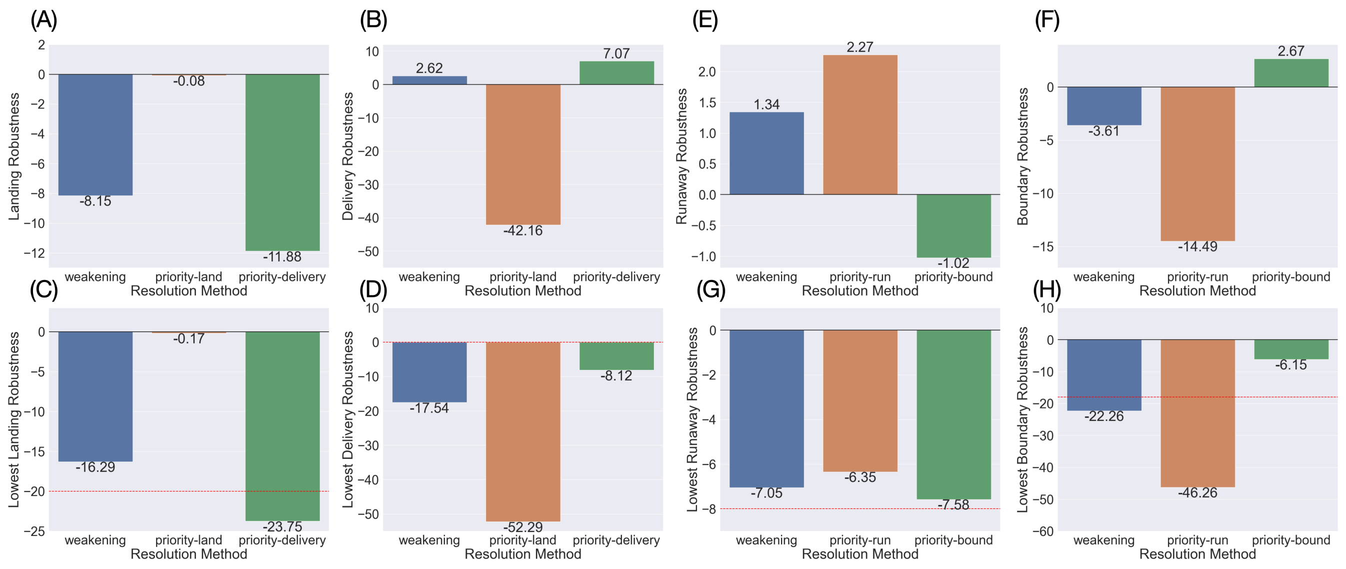

Fig. 5: Robustness breakdown by each feature requirement. Chart groups (A)-(D) and (E)-(H) correspond to the organ delivery and surveillance drone case studies, respectively. Each of (A), (B), (E), and (F) compares the average robustness values for three different approaches: (1) weakening-based, (2) priority-based with feature 1, and (3) priority with feature 2. Charts (C), (D), (G), and (H) show the lowest robustness values; the red line represents the threshold at which the minimal requirement is violated.

(E) and (F), it can be seen that on average, the weakening-based approach achieves a compromise between prioritizing vs discarding a feature as it is done by the priority-based method.

In (H), for the boundary enforcer, the weakening-based approach is not able to guarantee the minimal requirement, violating it in its worst-case outcome (-22.26). This is due to the fact that occasionally, the feature interaction scenario forces the drone into a non-recoverable position (i.e., cannot avoid crashing into the boundary), causing the resulting MILP problem to be unsatisfiable. Even then, it can be seen that the weakening-based approach avoids the large violation that is caused by the priority-based method (-46.26).

3) Summary: Based on Fig. 4, we conclude that the weakening-based approach attains a higher overall robustness than the priority-based approach, supporting hypothesis **H1**. This suggests that the weakening-based approach is effective at satisficing the requirements of both conflicting features.

In Fig. 5, (C-D), (G-H), it can be seen that the weakening-based approach, in its worst-case outcome, may fail to guarantee the minimal requirement under environmental scenarios that does not permit any weakening solution. This suggests that if satisfying a particular requirement is critical, the priority-based method that always selects that feature may be more desirable. Thus, hypothesis **H2** is not supported.

E. Performance Overhead

Since the weakening-based approach uses a MILP solver, it incurs significantly more overhead than the priority-based method, which simply involves selecting the preferred action. Based on our timing measurement, the MILP-based resolution process took approximately 0.28 seconds on average across the two case studies. During our simulation runs, we did not observe any noticeable delays or disruptions to the drone operation. This is because the control loop inside the PX4 drone software was running at 2Hz, resulting in a window of 0.5 seconds for each cycle of the controller update (i.e., the resolution process completed before the next control action was to be generated).

On the other hand, the amount of overhead depends on the complexity of the feature requirements and the environment model, and thus it is possible that one may run into performance issues for larger, more complex systems than our drone software. As part of future work, we plan to explore other methods that leverage different types of search heuristics (e.g., machine learning-based or generic algorithms) and could potentially provide more efficient resolution.

F. Threats to Validity

There are three sources of potential errors in our experiments: (1) the selected case study may not be representative of general CPS applications, (2) the scenarios generated may exclude exceptional scenarios, and (3) drone simulation is hardware-dependent.

For (1), we believe that the two case studies we conducted embody common characteristics of CPS, as PX4 is a well-established, popular drone software. However, a more extensive validation that involves other types of CPS (e.g., automotive systems) may provide further support for the effectiveness of the proposed resolution approach. To address (2), we generated a comprehensive sampling of scenarios and reduced selection biases by randomizing the configuration parameters. However, our sampling is non-exhaustive and is likely to exclude some exceptional scenarios. For (3), more restrictive hardware may cause an increase in the performance overhead due to the computing resources that are required for the MILP solver and the simulator.

VIII. Related Work

There is a large body of work on feature interactions within software engineering [1]–[3], [7], [12], [27]–[34]. Here, we mainly provide a discussion of related work on resolution (rather than detection) of feature interactions.

Gafford et al. [26] proposes a *synthesis-based* approach to resolution of feature interactions, where given a pair of conflicting actions, they a space of possible alternative actions are enumerated to find an action that best satisfies the objectives of the two features. Their approach is similar to ours in that it also (1) relies on the notion of robustness in STL to define the desirability of an action and (2) attempts to find an action as a middle-ground between the two conflicting actions. However, their approach is limited to cases where the features generate the same type of action (e.g., a pair of velocity vectors), whereas our approach does not suffer from this limitation and can be applied to features with different types of actions (e.g., landing vs. flying towards destination in the organ delivery).

Maia et al. investigates the problem of *defiant components*, where one or more local components, in trying to achieve their individual objectives, conflicts with a global system requirement [19]. They propose an approach called *cautious adaption* to dynamically modify the behavior of the local component and fulfill the global requirement when a conflict arises; adaptation here is carried out by injecting a piece of logic (through aspect-oriented wrappers) into the local component that overrides its default behavior during a conflict. Although their approach shares some similarity with ours in that they both involve temporarily changing the objective of a system component, there are also some noticeable differences: (1) our work deals with conflicts between competing features (or components) instead of local vs. global requirements and (2) their approach requires wrappers that are crafted at design time to handle known "exceptional situations" (i.e., conflicts), whereas our approach can handle *unexpected* conflict scenarios, as long as they are captured by the environment model.

Requirement relaxation (or weakening) has been investigated in the context of self-adaptive systems. RELAX [35] is a temporal logic to support specification of requirements that explicitly capture uncertainty about possible system behavior. RELAX can be used to support self-adaptation mechanisms where the system dynamically adjusts its behavior to accommodate for uncertainty or changes in the environment. DeVries et al. uses RELAX to investigate the concept of *partial* feature interactions, where a pair of features, in presence of each other, only partially satisfy (i.e., satisfice) their individual requirements [36], although this work does not discuss a mechanism for resolving such interactions. One interesting future direction that we plan to explore is to leverage RELAX as another type of requirements specification language (instead of STL) to support weakening-based resolution.

In [37], the authors propose an iterative, *multi-grained* approach to requirements relaxation as part of a self-adaptation framework, where requirements of a higher granularity are relaxed first (for computational efficiency) before relaxing lower-level requirements. Although our approach currently assumes feature requirements to be at the same level of granularity, their approach may be useful for resolving more complex types of interactions that involve requirements across different levels of system abstraction.

Our approach for leveraging an environment model to generate an action that satisfies a desired objective (i.e., satisfaction of weakened requirements) can be regarded as a type of model-predictive control (MPC) [38]. In particular, we adopt the STL-based MPC method developed by Raman et al. [39], where they also leverage an MILP solver to synthesize an action that satisfies an STL property.

IX. Limitations and Future Work

We have proposed a reconciliation-based approach to resolution of feature interactions, where one or more of the given feature requirements are weakened in order to enable the conflicting features to behave consistently. Through case studies on autonomous drones, we have demonstrated that the proposed approach can achieve an overall higher satisfaction of the conflicting features, compared to the conventional priority-based method where only one of the features is selected.

Our work makes several assumptions about the characteristics of the underlying system. First, our approach is most effective for systems where requirements can be assigned a meaningful, quantitative notion of satisfaction (i.e., STL-based requirements) and where the system operator or user may be willing to accept temporary degradation in the system performance. Thus, this approach may not be suited for features that perform a very critical function (e.g., emergency braking feature in a vehicle) and where even small degradation is unacceptable; in such cases, a priority-based method that favors those features over the others may be more suitable.

Our approach also relies on a model of the environment that can be used to generate predictive signals for different feature actions. Instead of a deterministic, discrete-step transition system, some type of stochastic model (e.g., Markov decision processes) may provide a more realistic model of the environment. At the same time, such a model would also require a very different approach to weakening than our MILP-based method and is beyond the scope of this paper.

So far, we have evaluated the proposed resolution approach on pairwise feature interactions. Although, in principle, the weakening-based approach should be applicable to any number of features, a more extensive validation involving N-way interactions [40] would further support the generality of the proposed approach.

Finally, we believe that the idea of adapting system behaviors through weakening of requirements have other applications beside feature interaction resolution (e.g., using weakening to gracefully degrade the quality of a service in response to a violation of an environmental assumption). We plan to explore such applications as part of future work.

References

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

[2] A. Nhlabatsi, R. Laney, and B. Nuseibeh, "Feature interaction: The security threat from within software systems," *Progress in Informatics*, vol. 5, pp. 75–89, 2008.

[3] P. Zave, "Feature interactions and formal specifications in telecommunications," *IEEE Computer*, vol. 26, no. 8, pp. 20–30, 1993.

[4] L. Yarosh and P. Zave, "Locked or not?: Mental models of IoT feature interaction," in *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems, Denver, CO, USA, May 06-11, 2017.*, 2017, pp. 2993–2997.

[5] A. L. J. Dominguez, N. A. Day, and J. J. Joyce, "Modelling feature interactions in the automotive domain," in *International Workshop on Modeling in Software Engineering (MiSE)*, 2008, pp. 45–50.

[6] A. Metzger, "Feature interactions in embedded control systems," *Computer Networks*, vol. 45, no. 5, pp. 625–644, 2004.

[7] M. H. Zibaeenejad, C. Zhang, and J. M. Atlee, "Continuous variable-specific resolutions of feature interactions," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, September 4-8, 2017*, 2017, pp. 408–418.

[8] Y. Chen, S. Lafortune, and F. Lin, "Resolving feature interactions using modular supervisory control with priorities," in *Feature Interactions in Telecommunications Networks IV, June 17-19, 1997, Montréal, Canada*, 1997, pp. 108–122.

[9] J. D. Hay and J. M. Atlee, "Composing features and resolving interactions," in *ACM SIGSOFT Symposium on Foundations of Software Engineering, an Diego, California, USA, November 6-10, 2000, Proceedings*, 2000, pp. 110–119.

[10] P. A. Zimmer and J. M. Atlee, "Ordering features by category," *Journal of Systems and Software*, vol. 85, no. 8, pp. 1782–1800, 2012. [Online]. Available: https://doi.org/10.1016/j.jss.2012.03.025

[11] A. Chavan, L. Yang, K. Ramachandran, and W. H. Leung, "Resolving feature interaction with precedence lists in the feature language extensions," in *Feature Interactions in Software and Communication Systems IX, International Co nference on Feature Interactions in Software and Communication Systems, ICFI 2007, 3-5 September 2007, Grenoble, France*, 2007, pp. 114–128.

[12] C. Bocovich and J. M. Atlee, "Variable-specific resolutions for feature interactions," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, (FSE-22), Hong Kong, China, November 16 - 22, 2014*, 2014, pp. 553–563.

[13] S. G. Raghavan, K. Watanabe, E. Kang, C. Lin, Z. Jiang, and S. Shiraishi, "Property-driven runtime resolution of feature interactions," in *Runtime Verification - 18th International Conference, RV 2018, Limassol, Cyprus, November 10-13, 2018, Proceedings*, 2018, pp. 316–333.

[14] D. L. Parnas and J. Madey, "Functional documents for computer systems," *Sci. Comput. Program.*, vol. 25, no. 1, pp. 41–61, 1995.

[15] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*. Springer Berlin Heidelberg, 2004, pp. 152–166.

[16] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, 1982.

[17] Dronecode Project, "PX4 autopilot," https://px4.io, 2020.

[18] B. H. C. Cheng, R. de Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, G. D. M. Serugendo, S. Dustdar, A. Finkelstein, C. Gacek, K. Geihs, V. Grassi, G. Karsai, H. M. Kienle, J. Kramer, M. Litoiu, S. Malek, R. Mirandola, H. A. Müller, S. Park, M. Shaw, M. Tichy, M. Tivoli, D. Weyns, and J. Whittle, "Software engineering for self-adaptive systems: A research roadmap," in *Dagstuhl Seminar Report*, 2009, pp. 1–26.

[19] P. H. Maia, L. Vieira, M. Chagas, Y. Yu, A. Zisman, and B. Nuseibeh, "Cautious adaptation of defiant components," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2019, pp. 974–985.

[20] H. A. Simon, "Rational choice and the structure of the environment," *Psychological Review*, vol. 63, no. 2, pp. 129–138, 1956.

[21] A. Pnueli, "The temporal logic of programs," in *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, 1977, pp. 46–57.

[22] N. Nethercote, P. J. Stuckey, R. Becket, S. Brand, G. J. Duck, and G. Tack, "Minizinc: Towards a standard cp modelling language," in *Principles and Practice of Constraint Programming (CP)*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 529–543.

[23] V. Raman, A. Donzé, M. Maasoumy, R. M. Murray, A. Sangiovanni-Vincentelli, and S. A. Seshia, "Model predictive control with signal temporal logic specifications," in *53rd IEEE Conference on Decision and Control*, 2014, pp. 81–87.

[24] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2022. [Online]. Available: https://www.gurobi.com

[25] L. Meier, D. Honegger, and M. Pollefeys, "Px4: A node-based multithreaded open source robotics framework for deeply embedded platforms," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 6235–6240.

[26] B. Gafford, T. Dürschmid, G. A. Moreno, and E. Kang, "Synthesis-based resolution of feature interactions in cyber-physical systems," in *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2020, pp. 1090–1102.

[27] N. Siegmund, S. S. Kolesnikov, C. Kästner, S. Apel, D. S. Batory, M. Rosenmüller, and G. Saake, "Predicting performance via automated feature-interaction detection," in *34th International Conference on Software Engineering, ICSE 2012, June 2-9, 2012, Zurich, Switzerland*, 2012, pp. 167–177.

[28] S. Apel, H. Speidel, P. Wendler, A. von Rhein, and D. Beyer, "Detection of feature interactions using feature-aware verification," in *26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011), Lawrence, KS, USA, November 6-10, 2011*, 2011, pp. 372–375.

[29] S. Apel, W. Scholz, C. Lengauer, and C. Kästner, "Detecting dependences and interactions in feature-oriented design," in *IEEE 21st International Symposium on Software Reliability Engineering, ISSRE 2010, San Jose, CA, USA, 1-4 November 2010*, 2010, pp. 161–170.

[30] S. Apel, A. von Rhein, P. Wendler, A. Größlinger, and D. Beyer, "Strategies for product-line verification: case studies and experiments," in *35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013*, 2013, pp. 482–491.

[31] W. Scholz, T. Thüm, S. Apel, and C. Lengauer, "Automatic detection of feature interactions using the java modeling language: an experience report," in *Software Product Lines - 15th International Conference, SPLC 2011, Munich, Germany, August 22-26, 2011. Workshop Proceedings (Volume 2)*, 2011, p. 7.

[32] A. Classen, P. Heymans, P. Schobbens, A. Legay, and J. Raskin, "Model checking lots of systems: efficient verification of temporal properties in software product lines," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1, ICSE 2010, Cape Town, South Africa, 1-8 May 2010*, 2010, pp. 335–344.

[33] J. M. Atlee, U. Fahrenberg, and A. Legay, "Measuring behaviour interactions between product-line features," in *3rd IEEE/ACM FME Workshop on Formal Methods in Software Engineering, FormaliSE 2015, Florence, Italy, May 18, 2015*, 2015, pp. 20–25.

[34] 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.

[35] J. Whittle, P. Sawyer, N. Bencomo, B. Cheng, and J.-M. Bruel, "RELAX: A language to address uncertainty in self-adaptive systems requirement," *Requir. Eng.*, vol. 15, pp. 177–196, 06 2010.

[36] B. DeVries and B. H. C. Cheng, "Towards the detection of partial feature interactions," in *Proceedings of the 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems SEAMS*. ACM, 2019, pp. 146–152.

[37] J. Li and K. Tei, "Done is better than perfect: Iterative adaptation via multi-grained requirement relaxation," in *IEEE International Conference on Requirements Engineering (RE)*, 2022.

[38] E. F. Camacho and C. B. Alba, *Model predictive control*. Springer science & business media, 2013.

[39] V. Raman, A. Donzé, M. Maasoumy, R. M. Murray, A. L. Sangiovanni-Vincentelli, and S. A. Seshia, "Model predictive control with signal temporal logic specifications," in *53rd IEEE Conference on Decision and Control, CDC 2014, Los Angeles, CA, USA, December 15-17, 2014*, 2014, pp. 81–87.

[40] B. DeVries and B. H. C. Cheng, "Run-time monitoring of self-adaptive systems to detect n-way feature interactions and their causes," in *Proceedings of the 13th International Conference on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, 2018, pp. 94–100.